Fashion classification and segmentation

An instance segmentation approach

MSc Business Analytics Thesis

Author:

Katrin Kostova

Supervisor Vrije Universiteit Amsterdam:

Prof. Dr. A. E. Eiben

Second reader Vrije Universiteit Amsterdam:

Dr. J. Berkhout

Supervisor Bright Cape:

I. Brantevica

Fashion classification and segmentation

An instance segmentation approach

Public

Date: 7 August 2020
Author: Katrin Kostova
katrin@kostova.nl

Vrije Universiteit Amsterdam
Faculty of Science
Business Analytics
De Boelelaan 1081a
1081 HV Amsterdam

Bright Cape B.V.
Location Amsterdam
Startup Village
Science Park 608-K08
1098 XH Amsterdam

Preface

This thesis report is the end product of a six-month graduation internship of the Master program Business Analytics at the Vrije Universiteit Amsterdam. The corresponding specialization is Computational Intelligence. It is meant to present the findings and research that had been conducted in the field of computer vision, specifically image segmentation and multi-label image classification in the fashion industry. Even though the techniques used are closely related to the course Advanced Machine Learning, the internship relies heavily on the conducted literature review and require investment in new scientific areas.

The internship was offered by the company Bright Cape and took place at the Analytics & Applied Data Science department. Bright Cape is a data consultancy company based in Amsterdam, Eindhoven, and London. Bright Cape was founded at the end of 2014 and has since grown rapidly to 90+ employees ranging from data analysts, engineers and scientists, but also psychologists, user researchers, designers and developers [Bright Cape, nd].

This graduation internship is a part of an EIT Digital funded innovation called Customer Profiling & Recommendation system (CP&R). CP&R is a collaboration between Bright Cape, University of Rennes, and University of Barcelona. The research problem is to be able to classify and segment the clothing items that a person is wearing, based on a picture that is taken of them in the real life environment. The client of the project is a multinational operating mostly in South America.

I would like to thank my Vrije Universiteit Amsterdam supervisor Guszti Eiben for sharing his advice and facilitating a peer group in which we discussed each other's issues and best ideas. I would also like to thank my second reader Joost Berkhout for his time.

A huge thank you goes to my external supervisor Ieva Brantevica for the weekly calls, feedback on my writing and guidance and support throughout the whole internship and to Bright Cape for offering the internship as well as providing data, facilities and guidance. It was a great experience to work on this innovative project and gain hands-on experience in deep learning and computer vision.

Finally, I thank family and boyfriend for proofreading this report and for supporting me throughout the project. Mostly I am thankful that I was able to conduct the internship in good health with no delay caused by the COVID-19 pandemic.

Figure 1.: Sunset at Vrije Universiteit Amsterdam

Management Summary

Problem definition - The goal of this research is to choose and apply a computer vision model for high-level fashion segmentation. This model needs to be able to classify and segment the clothing items that a person is wearing based on a picture that is taken of them. For a model to have potential in real-life applications it needs to have short inference time using only CPU computing power. The main research question is 'Which deep learning algorithm performs best on the fashion classification and segmentation task?'. We define best as the trade-off between the quality of classification and segmentation and the average inference time.
Academic and Practical Relevance - We review the computer vision approaches multi-label image classification, object detection, semantic and instance segmentation, and the metrics used to evaluate them. We implement semantic and instance segmentation using the deep learning architectures RefineNet and Mask R-CNN. We optimize the accuracy while keeping the speed under 2 seconds ensuring real-life applications. The main practical challenge is the inference time, using only CPU computing power. We present the novel Trade-off metric by which we evaluate the performance of the implemented models.
Methodology - Using the publicly available iMaterialist dataset we experiment with training different models using the two model architectures. The data is pre-processed to speed up the model's convergence, improve the learning process, and compensate for the real-life image quality. Transfer learning is used to help overcome the model's bias towards training data, learning faster, and prevent overfitting. The algorithms are tuned using a grid search through the extensive list of parameters belonging to each of the architectures.
Results - The two algorithms are inferred on both a set of images provided by the client of the project and a test set from the iMaterialist dataset. We recognize that given the limited resources, a lack of pre-trained weights and possibly the fact that semantic segmentation is not the best approach for the task at hand, our experiments with RefineNet are fruitless. Contrastingly, the instance segmentation architecture Mask R-CNN shows robust results in both classification and segmentation. Moreover, we manage to get the average inference time of the best model to 1.37 seconds by reducing the image size and number of detection instances.
Recommendations - The resulting model can be implemented directly or in combination with a post-processing scheme. The suggestions for further research are to include training images of population minorities, include training images of traditional cultural clothing, and potentially extract fabric information from the resulting segments.

Contents

List of Abbreviations

Adam	Adaptive Moment Estimation. 39
AI	Artificial Intelligence. 1, 15, 59
AP	Average precision. 13
AWS	Amazon Web Services. 31, 52
CNN	Convolutional Neural Network. 7, 9, 17, 19, 20
COCO	Common Objects in COntext. xi, 21, 27, 29, 31, 32, 34, 47
CP&R	Customer Profiling & Recommendation system. xi, xiii, 1–3, 13, 34–36, 52
CPU	Central Processing Unit. 20, 31, 34, 47, 51
DSC	Dice similarity coefficient. 12
EMR	Exact Match Ratio. 10–12, 14, 35, 37, 40–42
Faster R-CNN	Faster Region-Based Convolutional Neural Networks. 7–9, 21, 22
FCN	Fully Convolutional Network. 18, 20, 22
FN	False negative. 12
FP	False positive. 12
GPU	Graphics Processing Unit. 9, 20, 31, 33, 34, 47
IoU	Intersection over Union. 7, 8, 11–13
JSS	Jaccard similarity score. 11, 12, 14, 35, 37, 40, 41
mAP	Mean Average Precision. 11–14, 20, 35, 38, 40, 41
OA	Overall accuracy. 7, 10–12, 14, 35, 37, 40, 41

R-CNN Regions with Convolutional Neural Network features. 19–21

R-FCN Region-Based Fully Convolutional Networks. 8

RCU Residual convolutional unit. 19

RefineNet Multi-path Refinement Network. v, vi, 8, 14, 15, 18, 19, 35, 38, 45, 47, 48

ResNet Residual Net. 18, 19, 33, 46, 47

RGB Red Green Blue, three-channel additive color model. 20, 23, 26, 29, 46, 51

RNN Recurrent Neural Network. 7

RoI Region of Interest. 15, 22, 33, 39

RoIAlign Region of Interests Align. 9, 21, 22

RPN Region Proposal Network. xi, 8, 20, 21

SGD Stochastic Gradient Descent. 17, 33, 39

SSD Single Shot MultiBox Detector. 8

SVM Support Vector Machine. 20

TO Trade-Off score. 35, 38

TP True positive. 12

TTO Total Trade-Off score. 35, 38, 40–42

YOLO You Only Look Once. 8

List of Figures

List of Tables

1. Introduction

1.1. Motivation for the research

Revenue in the Fashion segment in the Netherlands amounts to €4.736m in 2020 and is expected to show a growth of 8.3% in 2021 [Statista, 2020]. However, the Netherlands is in Europe's top three when it comes to online retail. The Dutch B2C e-commerce was valued at €23.7 billion in 2018 and increasing [de Best, 2019]. This translates to the competition getting harder each year for the physical stores. The closing doors of Intertoys, V&D and the short stay of Hudson's Bay in the Netherlands are one of many examples of stores failing to adapt and innovate in time. To be able to keep up with the online experience, the physical retailers need to up their game. That could be realized by innovating and making their stores more technical, exciting and competent in assisting the customer. The stores, on the other hand, need more data about the people that enter the door to be able to achieve accurate customer profiling. The profiling can be based on the characteristics such as age, gender and clothing choices of their customers and can be used for targeting the marketing and increasing sales.

This research fills the gap between the industry's demand for generating data for customer profiling and the customer's need for an exciting shopping experience. This innovation project is a next-generation CP&R, using deep learning. CP&R is a smart system that automatically creates customer profiling by using cutting-edge technology on artificial intelligence and neuromarketing. The solution is going to be implemented into omnichannel marketing, the sales approach that provides the customer with an integrated shopping experience. Moreover, this solution integrates knowledge from an interdisciplinary team of experts from neuromarketing, Neuro-Linguistic Programming, and Artificial Intelligence (AI) fields.

1.2. Scope

The CP&R project consists of two phases. Phase one is the customer profiling that consists of developing and training an object detection algorithm that detects the person on the image, classifying their gender, age category and classifying and segmenting the present apparel. This first phase is the task of Bright Cape. Next, phase two is building a recommender system based on the extracted characteristics that is the contribution of the University of Barcelona.

This thesis research is a part of phase one, namely fashion classification and segmentation. The gender and age classification has already been delivered. Also, the recommender system is out of scope as it is the second step of the project and needs the output of the classifications as input. This research focuses on delivering a computer vision algorithm of high-level apparel classification and segmentation. See Figure 2 for a visual clarification and the positioning of this research. As can be seen in this figure the products of the CP&R project are a fashion recommendation for the customer and customer profiling and analytics for the client.

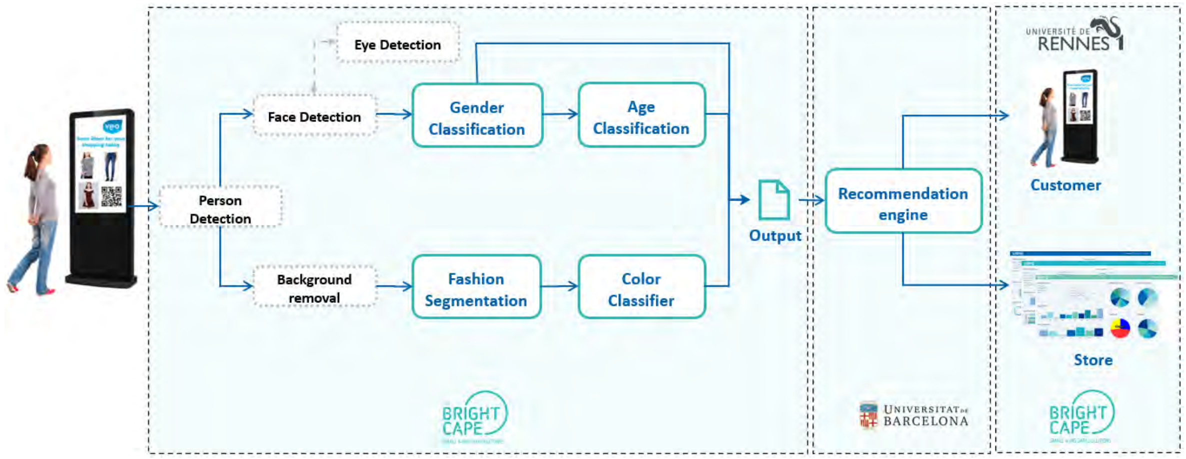


Figure 2.: Overview of the CP&R project phases and underlying components and products

1.3. Research objective

The goal of the internship is to choose and apply a computer vision model for high-level fashion segmentation. This model needs to be able to classify and segment the clothing items that a person is wearing based on a picture that is taken of them. Examples of fashion items are shoes, dresses, top/t-shirt/sweatshirts, pants, and jackets. After deployment of the model, the results are to be obtained on demand using a standard laptop. This is why for a model to be applicable it needs to be fast. We aim to optimize the quality of classification and segmentation while keeping the inference speed under two seconds. The proposed task unifies both classification and segmentation of rich and complete apparel attributes, an important step toward real-world applications. The main research question is as follows:

Which deep learning algorithm performs best* on the fashion classification and segmentation task?

This research question is decomposed in the following sub-questions:

1. Which computer vision approaches are best suited for the task at hand and what are the corresponding, most prominent methods for implementation?

2. Which and how much data is needed to train the high-level segmentation models and how should this data be pre-processed?
3. Which data should be logged while training and testing in order to measure the performance of the models?
4. Is the complexity of a model architecture related to its performance in terms of the classification and segmentation quality and inference time? If yes, how?
5. Is inference time of under one second for classifying and segmenting the fashion items of one image achievable?
6. Which model performs best* on the iMaterialist test set and Client dataset based on the chosen performance measures?

* Best is defined as a trade-off between the quality of classification and segmentation and the inference time. The performance testing metrics are discussed in Section 2.3.

1.4. Approach

Computer vision is the construction of explicit, meaningful descriptions of physical objects from images by computers [Ballard and Brown, 1982]. The approach is oriented towards deep learning and image segmentation. The available pixel annotated dataset is the iMaterialist dataset (see an example in Figure 3) [iMaterialist Fashion Competition group, 2019]. We use

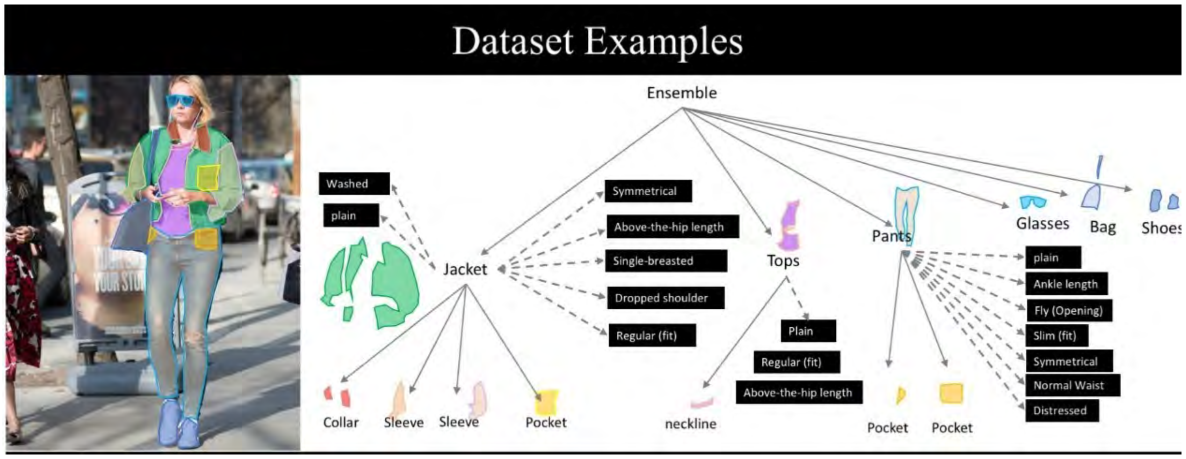


Figure 3.: Sample of the iMaterialist dataset

the majority of the dataset for training and validation of the models and a small portion for inference. Furthermore, an image annotated set has been provided by the client of the CP&R project that is strictly used for inference.

The steps taken toward the final model are initially training and testing each of the two selected architectures on a small subset of five base classes being 'shoe', 'dress', 'top, t-shirt, sweatshirt', 'pants' and 'jacket'. The initial experiments provide an indication on the capability of the architecture to learn the characteristics of the fashion items and the time it takes to infer one image. Next, we select an extended list of eleven classes being 'shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'jacket', 'pants', 'skirt', 'dress', 'tie', 'shoe', 'bag, wallet' and 'umbrella', and train and tune the final model on it.

1.5. Report structure

This report continues with providing the conducted literature review in terms of computer vision approaches, performance testing, suited methods and a conclusion. Then, the two chosen algorithms are described in terms of their architecture and hyperparameters in Chapter 3. This is followed by the description of the experimental setup in Chapter 4 where the problem instances, data pre-processing steps and training details. Chapter 5 is where the results of both algorithms are provided in terms of inference time and performance quality. The answers to the sub-questions presented in the introduction are provided in Chapter 6 next to a discussion of the results. Finally, the answer to the main question is provided in Chapter 7.

2. Related Work

In this chapter we show the outcome of the literature review that gave basis for the performed experiments and their setup. In Section 2.1 we describe the scientific background and discuss which computer vision approaches are suited for the task at hand. Out of the nine introduced approaches, four are chosen. These four suited approaches are described together with their respective state-of-the-art research method in Section 2.2. As we are handling image data it is not self-explanatory which performance measures to use. The overview of commonly used performance measures is given in Section 2.3. We wrap up the literature review by summarizing the learned lessons in Section 2.4.

2.1. Computer vision

This is a computer vision task. The goal is scoped to the two-dimensional imaging-plane as the data we aim to classify are images. This is important to be noted as computer vision also includes techniques for handling 3D data with depth perception. There is a second division made in computer vision, that is the dichotomy between recognising stuff and things. Things are countable objects such as people, animals and tools. Stuff on the other hand are amorphous regions of similar texture or material such as grass, sky and road. Recognizing things is typically the task of image classification, object detection and instance segmentation. Studying stuff is most commonly formulated as a task known as semantic segmentation [Adelson, 2001].

2.1.1. Considered approaches

The available iMaterialist fashion dataset [iMaterialist Fashion Competition group, 2019] is pixel annotated, hence is it possible to train an algorithm to assign labels on pixel level. However, in the spirit of the principle of parsimony also known as Occam's razor we also consider two simpler approaches, classification and detection [Thorburn, 1915]. The top four suited techniques are:

- multi-label image classification;
- object detection;
- semantic segmentation;
- instance segmentation.

We omitted some techniques that are not directly applicable for the fashion apparel prediction. For example, single-label image recognition, object tracking – usually used for video and real-world interactions, image reconstruction – usually used for restoring old or incomplete images, multi-task learning and panoptic segmentation [Kirillov et al., 2019].

Both multi-task learning and panoptic segmentation encompass both stuff and thing classes in a single framework. The disadvantage of multi-task learning is that the output of the stuff classes is independent from the thing classes and possibly inconsistent. Panoptic segmentation on the other hand requires a single coherent scene segmentation and takes into account the dependency between the classes. Panoptic segmentation sounds very promising but is omitted, because it may overcomplicate this research and is still not yet well studied.

2.1.2. Complexity

The four approaches are listed in order of the complexity of the problem they solve. Larger networks achieve higher accuracy on more complex tasks. This is why we quantify the complexity of an algorithm by its size. The number of layers and number of neurons in a neural network are called depth and width respectively. Both the depth and width can be used as a metric for neural network size [Goodfellow et al., 2016]. For instance the Percepton is a network with a single layer and a single neuron [Rosenblatt, 1957], thus having both depth and width equal to one.

2.2. Suited approaches

The approaches that are suited for executing the task at hand are multi-label image classification, object-detection, semantic segmentation and instance segmentation. See Figure 4 for a visual example [Harper, 2019].

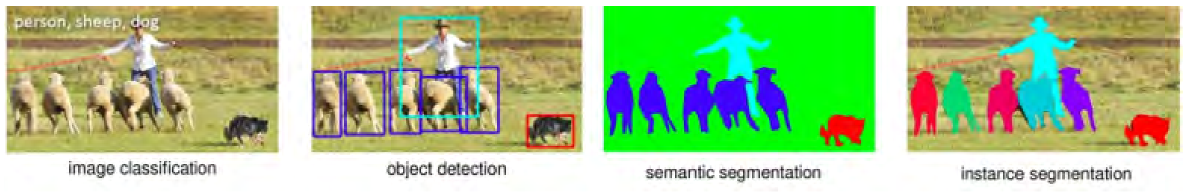


Figure 4.: The four candidate computer vision techniques

In this section we introduce these approaches and sketch the idea behind methods used to perform them. We present one method per approach in order to keep this section compact even though we have studied papers about the performance of many other methods.

2.2.1. Multi-label image classification

This approach stems out of the classical single-object image classification but takes into account that one observation can belong to multiple classes at the same time. This is not to be confused with multi-class classification where there are more than two classes, but the observation belongs to strictly one class. Possible applications for multi-label image classification are complex medical and technical images [Nair, 2019]. For instance, guessing the content of a movie by its poster. The data structure needed for this task is an image per movie that is labelled with one of the ground truth labels that are in this case a combination of the genres animation, comedy, romance, history etc. Other features can be title, IMDb score, cast etc. As the prediction is per-image, it is possible to use conventional quality measurements as accuracy and precision, only now it can be split into per class accuracy and the Overall accuracy (OA).

CNN-RNN is a framework based on deep Convolutional Neural Network (CNN), including the explicit modelling of the dependencies between multiple labels with recurrent neural networks Recurrent Neural Network (RNN). The RNNs framework is designed to adapt the image features based on the previous prediction results, by encoding the attention models implicitly in the CNN-RNN structure. Specifically, it first recognizes the dominant objects after which the focus is shifted to the smaller and harder to recognize ones. This provides enough context for the smaller objects to be easily inferred [Wang et al., 2016].

2.2.2. Object detection

Object detection, also called bounding-box detection method, is a method where each object is represented by a crude rectangular box. This approach adds the element of localization of an object next to classifying it. It can be used in self-driving cars, surveillance and quality control of parts in manufacturing etc. For instance, detecting blood cells via microscopic image reading to gain information about the immune system of a person. The train data consists of the image id, ground truth classes and bounding box coordinates per object. There can be multiple rows for one image as each row represents an object of interest. This approach needs a different prediction measure than accuracy as we are interested in not only the predicted class of the object, but also the tightness and location of the bounding box. The most popular measure for this task is Intersection over Union (IoU) that compares the area of intersection with the area of the union between the actual and the predicted bounding box.

Faster Region-Based Convolutional Neural Networks (Faster R-CNN) [Ren et al., 2015] is a canonical model for deep learning-based object detection. In its core it is a Fast R-CNN based algorithm [Girshick, 2015], which again is based on R-CNN [Girshick et al., 2014], where the selective search algorithm used to generate region proposals is replaced by a fast neural network by inserting a RPN. The RPN predicts the regions where further processing needs to occur in order to reduce the computational requirements of the overall inference process. Faster R-CNN is the state-of-the-art model in terms of performance, even though there are many new techniques as You Only Look Once (YOLO) [Redmon et al., 2016], Single Shot MultiBox Detector (SSD), and Region-Based Fully Convolutional Networks (R-FCN) [Le, 2018].

2.2.3. Semantic segmentation

Semantic segmentation is a segmentation approach focused on recognizing stuff. It entails classifying all the pixels in the image, not only the objects of interest. The datasets used for this kind of segmentation contain both stuff and thing classes, but do not distinguish individual object instances. For instance, material and texture will effect the classified images. It comes closer to the way a human would assess an object as we also distinguish between amorphous regions of similar texture such as snow from ice cream, silk from leather, pavement from mud and a flower print on a t-shirt from an actual flower. Therefore, unlike classification and object-detection, we need dense pixel-wise predictions from our models [Adelson, 2001]. Stuff classes are important as they allow to explain important aspects of an image, such as scene type, contextual reasoning, physical attributes and geometric properties of the scene. Segmentation task requires, again, a novel metric to measure the quality of the prediction. The most popular is the pixel accuracy that is the percentage of correctly labelled pixels in the dataset [Caesar et al., 2018] and mask IoU.

Most architectures used for semantic segmentation rely heavily on fully convolutional networks. That holds for instance for both RefineNet [Lin et al., 2017a] and DeepLab [Chen et al., 2016a] that are the popular and state-of-the-art networks for this task. RefineNet is a network which exploits features at multiple levels of abstraction for high-resolution semantic segmentation. RefineNet refines low-resolution (coarse) semantic features with fine-grained low-level features in a recursive manner to generate high-resolution semantic feature maps. This model is flexible in that it can be cascaded and modified in various ways. In comparison RefineNet is said to have shorter inference time and higher IoU on both Cityscapes test set [Cordts et al., 2016] and Person-Part dataset [Chen et al., 2016b].

2.2.4. Instance segmentation

Instance segmentation is the task of locating all objects in an image, assigning each object to a specific class and generating a pixel-perfect mask for it, delineating their shape. Instance segmentation is most popular for real-time applications like autonomous driving, robotics, analysing applications and is a very well researched approach. The advantage of this segmentation method in contrast to semantic segmentation is that different instances of the same class are segmented individually in instance segmentation, as can be seen in Figure 4. This would be the preferred approach in a task where the count of the same items is something that we are interested in. Methods for instance segmentation are either proposal-based or proposal-free approaches [Hsu et al., 2018]. The first one is the dominant one with the positive of it achieving outstanding results on many benchmarks and the negative of generating low resolution masks, which are not desirable for some applications. This approach entails detecting objects using a bounding-box detection method and then generating a binary mask for each one. The Mask R-CNN framework is the most used method for instance segmentation to date. The second approach for instance segmentation is mostly based on embedding loss functions or pixel affinity learning [Neven et al., 2019]. The pros of proposal-free approach is that the instance masks can have high resolution and faster run-times, but fail to perform as well as the proposal-based approaches.

Mask R-CNN carries out pixel-level segmentation by taking the predicted class and bounding box coordinates for each object that is predicted by Faster R-CNN and adding a mask branch to the architecture. The branch is a Fully Convolutional Network on top of a CNN-based feature map. Given the CNN Feature Map as input, the network outputs a binary mask using a method known as Region of Interests Align (RoIAlign). Essentially, RoIAlign uses bi-linear interpolation to avoid error in rounding, which causes inaccuracies in detection and segmentation. The authors report an inference time of 195 milliseconds using a NVIDIA Tesla M40 Graphics Processing Unit (GPU) [NotebookCheck, ndb] [Le, 2018].

2.3. Performance testing

In order to measure the performance of the proposed methods on the two available test sets we referred to the literature for the most common and useful measures. The two performance aspects that we are interested in are:

- Classification and segmentation quality:
  - Multi-label classification;
  - Image segmentation;
- Inference time in seconds.

For the quantification of these measurements we use the formal notations used throughout this section in Table 1.

Table 1.: Formal Notation

Notation	**Explanation**
C, M	Client dataset, iMaterialist dataset
$\lvert C\rvert = c$	total number of images in C
$\lvert M\rvert = n$	total number of images in M
$\lvert M_{test}\rvert = t$	total number of images in M_{test}
$\lvert S\rvert = s$	total number of segment masks
$\lvert O\rvert = o$	total number of objects
(h, w, c)	image size height, weight and channel count
S_i	segment mask of image i, size (h, w, o_i)
$\lvert S_i\rvert$	number of segment masks in image i
o_i	number of objects in image i
B_i	bounding box of image i, size $(o_i, 4)$
K	labelset
$\lvert K\rvert = k$	length of labelset
h	multi-label algorithm
$X_i = \{0,1\}^k$	set of label membership for the example x_i
$Y_i = \{0,1\}^k$	set of true label membership for the example x_i
$Z_i = h(x_i) = \{0,1\}^k$	set of label membership predicted by h for the example x_i
I	indicator function
$\lvert(X_i \cap Z_i)\rvert$	intersection between X_i, Z_i
$\lvert(Y_i \cap Z_i)\rvert$	intersection between Y_i, Z_i
$\lvert(Y_i \cup Z_i)\rvert$	union between Y_i, Z_i
r, p	recall levels, precision levels
L	loss function
T_{inf}	average inference time
τ	current time

2.3.1. Classification and segmentation quality

Above all the aim is to measure the quality of the classification and segmentation. As briefly mentioned in Subsection 2.1.2 the metrics to quantify this performance differ per approach. The literature stated that it is typical to document the results of an algorithm using multiple measures. This is why we select a number of measures for each task.

Evaluation of multi-label classification

Evaluation of multi-label classification algorithm has added complexity due to having an additional notion of being partially correct. The three measures that provide most insight into the quality of prediction in multi-label image classification are:

- Exact Match Ratio (EMR);
- Overall accuracy (OA);

- Jaccard similarity score (JSS).

Exact match ratio - This metric ignores partially correct instances and considers them incorrect. In an example where our model might predict nine fashion items correctly and one incorrectly, this image prediction will be counted for as incorrect using EMR. This is why this measure is considered harsh. The formula used to calculate EMR is shown in Equation (1) [Sorower, 2010]:

$$EMR = \frac{1}{n}\sum_{i=1}^{n} I(X_i = Z_i). \tag{1}$$

The overall accuracy - In order to account for the partially correct instances we consider the OA. Accuracy is the portion of the correct predicted labels divided by the total predicted labels per instance. OA is the average per class accuracy across all predicted images, calculated using

$$OA = \frac{1}{n \cdot k}\sum_{i=1}^{k} |(X_i \cap Z_i)|. \tag{2}$$

Jaccard similarity score - This metric calculates the ratio of intersection and union of the true and predicted labels. It is otherwise known as the IoU coefficient. The formula is shown below in Equation (3):

$$JSS = \frac{1}{n}\sum_{i=1}^{n} \frac{|(Y_i \cap Z_i)|}{|(Y_i \cup Z_i)|}. \tag{3}$$

Example - To illustrate the multi-label classification metrics consider an image ($n = 1$) with 5 classes ($k = 5$). The true and predicted occurrences and calculated classification metrics are shown in Table 2. The EMR is 0%, because the true and predicted labels do not match exactly. The OA is 80% calculated with

$$\frac{1}{1 \cdot 5}(0 + 1 + 1 + 1 + 1) = \frac{4}{5}. \tag{4}$$

The intersection divided by the union of the positive labels JSS is 75%

$$1 \cdot \frac{(0 + 1 + 1 + 0 + 1)}{(1 + 1 + 1 + 0 + 1)} = \frac{3}{4}. \tag{5}$$

Evaluation of pixel-wise segmentation

Evaluation of segmentation is frequently done using pixel accuracy, Mean Average Precision (mAP) and F1-score. Pixel accuracy calculates the correctly predicted pixels over the total amount of pixels [Long et al., 2015a]. However, it is not recommended when dealing with

Table 2.: Example of the multi-label classification metrics

	top	jacket	pants	dress	shoe	Metric
True	1	1	1	0	1	
Predicted	0	1	1	0	1	
True/False	✗	✓	✓	✓	✓	
EMR						0%
OA						80%
JSS						75%

unbalanced data. In an image with a tiny fashion item, say shoe, in it the pixel accuracy can be 96% while the prediction is only background and thus worthless. This is why we do not use this metric. The results shown in Chapter 5 are in terms of:

- mAP;
- F1-score.

To evaluate the segmentation using mAP and F1-score we need to compare each of the predicted masks with each of the ground truth masks. For this goal we use Equation (3) also known as the IoU to define:

- A True positive (TP) is observed when a prediction-target mask pair has an IoU score which exceeds some predefined threshold;
- A False positive (FP) indicates a predicted object mask that has no associated ground truth object mask;
- A False negative (FN) indicates a ground truth object mask that has no associated predicted object mask;

The commonly used threshold for this purpose is 0.5 [He et al., 2017]. Once we have the TP, FP and FN we can calculate the precision:

$$Precision = \frac{TP}{TP + FP}, \tag{6}$$

and recall:

$$Recall = \frac{TP}{TP + FN}. \tag{7}$$

F1-score - The F1-score is the harmonic mean of these two measures. The F1-score is also known as the Dice similarity coefficient (DSC). It is calculated by using Equations (6) and (7)

$$F1_{score} = \frac{1}{s}\sum_{i=1}^{s} 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{8}$$

mAP - First we calculate the IoU overlaps between two sets of masks using Equations (6) and (7). We use 0.5 as threshold and annotate the Average precision (AP) using a 50% mask IoU threshold with AP@50 [Girshick, 2015]. AP@50 is the area under the interpolated precision-recall curve [Guangyu Zeng, 2018]:

$$AP = \sum_{j=1}^{o} (r_{j+1} - r_j) \cdot p_{interp}(r_{j+1}). \tag{9}$$

The total mAP is the average of AP across all classes averaged over all images

$$mAP = \frac{1}{n \cdot k} \sum_{i=1}^{n} \sum_{j=1}^{k} AP@50_{ij}. \tag{10}$$

2.3.2. Inference time

Inference is the process of running data through a model to obtain predictions. The time it takes to infer is the inference time. This measure is defined as the difference between the start of inference τ_{start} and the moment of obtaining the predictions τ_{end} and is measured in seconds. It is then averaged over the number of examples in the test set t

$$T_{inf} = \frac{1}{t} \sum_{i=1}^{t} \tau_{start,i} - \tau_{end,i}. \tag{11}$$

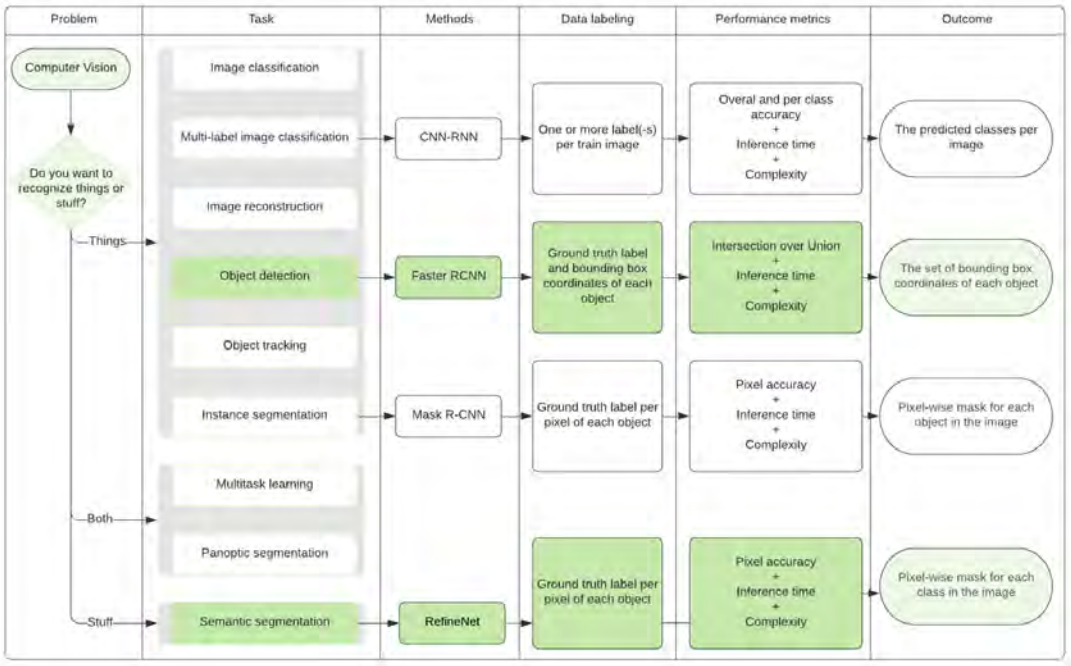
The average inference time is a key aspect of the CP&R project as customer profiling and recommendation need to be done on demand. We aim for the fastest possible model and rule out models slower than 2 seconds. We assume that the model weights and initialization are already done. This is a realistic assumption, as this process can be included in the start-up procedure, for instance each morning before opening.

2.4. Conclusion

This chapter aims to choose a direction in terms of a computer vision approach to proceed with, a method for implementation of that approach and the measures to test the performance with. In conclusion, after rigorous research, we showed that both recognizing things and stuff have numerous advantages. The applicable approaches have been sorted out and a final selection has been made that exists of semantic and instance segmentation. Segmentation produces a detailed mask of each object in the image instead of just the label or box around it what brings extra information about the shape. It takes advantage of the pixel-wise nature of the available dataset.

Semantic segmentation adds the aspect of recognizing stuff regions. In the task of predicting what a person is wearing from a picture given that people mostly wear layers of clothes and especially in the autumn and winter, the stuff approach seems to be a better fit. The visible part of the present apparel can seem amorphous when worn as an underlayer and/or photographed from a specific angle. Instance segmentation, on the other hand, identifies each segment and distinguishes objects of the same class, which can be useful for, for instance, detecting shoes.

The selected model architectures are RefineNet for semantic segmentation and Mask R-CNN for instance segmentation. We could find benchmarks for the inference time of the selected segmentation models, but those are based on inference using a professional graphical card. This contrasts with the computer used for the deployment of our model. Their documented segmentation performance on multiple datasets is state-of-the-art and very promising for our application. We choose to use the metrics EMR (1), OA (2) and JSS (3) for determining which objects are present in one image and F1-score (8) and mAP (10) for determining the quality of the segmentation. The chosen speed metric is the average inference speed measured in seconds (11).
See Figure 5 for a visual summary of the Literature review. The description of the applied algorithms follows in Chapter 3.



Problem
Task
Methods
Data labeling
Performance metrics
Outcome
Computer Vision
Do you want to recognize things or stuff?
Things
Both
Stuff
Image classification
Multi-label image classification
Image reconstruction
Object detection
Object tracking
Instance segmentation
Multitask learning
Panoptic segmentation
Semantic segmentation
CNN-RNN
Faster RCNN
Mask R-CNN
RefineNet
One or more label(-s) per train image
Ground truth label and bounding box coordinates of each object
Ground truth label per pixel of each object
Ground truth label per pixel of each object
Overal and per class accuracy + Inference time + Complexity
Intersection over Union + Inference time + Complexity
Pixel accuracy + Inference time + Complexity
Pixel accuracy + Inference time + Complexity
The predicted classes per image
The set of bounding box coordinates of each object
Pixel-wise mask for each object in the image
Pixel-wise mask for each class in the image

Figure 5.: Computer vision: Problem, Task, Methods, Data labelling, Performance metrics and Outcome

3. Algorithm Description

In this chapter we provide the explanation of the common mechanisms in Section 3.1, followed by the implemented algorithms RefineNet and Mask R-CNN in respectively Section 3.2 and 3.3. These are the two selected architectures that were chosen after a rigorous literature review. The main difference between the two algorithms is that RefineNet performs per-pixel multi-class categorization which couples segmentation and classification and Mask R-CNN predicts a binary mask for each class independently and rely on the network's Region of Interest (RoI) classification branch to assign a label. Formally RefineNet performs semantic whereas Mask R-CNN performs instance segmentation. In addition to the notations from Table 1, we define algorithm notations in Table 3.

Table 3.: Formal Notation: Algorithm description

Notation	**Explanation**
θ	parameters
τ	epoch
α	learning rate
x	examples of inputs
y	examples of outputs
d	depth of network

3.1. Common mechanisms

In the early days of the thriving field of AI, the first problems to get solved are the intellectually challenging for humans, mathematical rule based problems as these are the easiest for a computer. The ones that are easy for people, on the other hand, appear to be the hardest for machines. Those are challenging to describe formally and solved intuitively by humans, for instance, recognizing voices and speech and visually analyzing the context of a scene. These types of problems are being solved by gathering knowledge from experience and thus avoiding the need of hard-coding explicit rules to follow. The world is grasped by a hierarchy of simple concepts combined forming a deep network giving the name to the AI approach deep learning.

The capability of a system to acquire knowledge by extracting patterns from raw data is called machine learning. Logically, the representation of the data is a key element for this type of learning. The representation of information pieces is also known as features. For many tasks, it is difficult to know which set of features to extract.

If we take the simplified version of our task in which we want to determine the presence of a shirt in an image we might want to use the presence of a sleeve as a feature. Unfortunately, it is almost impossible to define a sleeve directly in terms of pixel values. Besides, there might be shadows, objects partially covering the sleeve, the silhouette depends on the viewing angle, etc., making the task even harder. Instead of defining the features manually, we use representation learning. This approach uses machine learning for both the representation itself and the mapping from representation to output. A figure showing the rule-based, classic machine learning and representation learning approaches and their respective hand-designed and learned components is shown in Appendix A. It appears that representation learning algorithms result in better performance as well as faster completion compared to hand-designed representations for complex tasks [Goodfellow et al., 2016].

Representing a sleeve directly from pixel values might be nearly impossible, but by using deep learning we can represent it in terms of a combination of simpler concepts such as corners, which are again a combination of even simpler concepts such as edges.

3.1.1. Deep learning background

The quintessential deep learning model is a feedforward neural network, also called a multi-layer perceptron. The goal is to approximate a function f^*. A feedforward network defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation. Feedforward networks form the basis of many applications, for instance convolutional neural networks. The network terminology comes from the fact that the architecture is a chain of functions. $f^{(1)}$ also called the first layer or input layer and $f^{(d)}$ called the output layer, with d being the depth of the network. The other layer(-s) are called hidden layers as the training data does not show the desired output for each of these layers.

Cost function - A practitioners goal is to optimize a performance metric P, but this can only be done indirectly while training by minimizing a cost function $J(\theta)$

$$J(\theta) = E_{(x,y)\ \hat{p}_{data}} L(f(x;\theta), y), \tag{12}$$

where L, x and $\hat{p}_{data}$ are the per-example loss function, inputs and empirical distribution respectively and $f(x,\theta)$ is the predicted output.

Optimization function - The Stochastic Gradient Descent (SGD) function updates the weights and biases θ at each epoch τ considering the learning rate α and loss function L

$$\theta(\tau + 1) = \theta(\tau) - \alpha \frac{\partial L}{\partial \theta}. \tag{13}$$

This optimization function as shown in Equation (13) updates all parameters with a constant learning rate at the end of each epoch. We make an improvement to this approach by decreasing the learning rate gradually according to a schedule. The learning rate at iteration e is denoted as a_e. The advantage to this is that when approaching the minimum we decrease the learning rate, making smaller updates to the weights. This helps avoid missing the minimum. Starting off with a very small learning rate is not recommended as the learning will proceed as slowly.

Regularization - The second most important concern for machine learning after optimization is regularization. If an algorithm generalized properly, the gap between the training and test error will be small. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error [Goodfellow et al., 2016].

3.1.2. Convolutional Neural Networks

CNNs are deep neural networks that use convolution instead of general matrix multiplication in at least one layer [Goodfellow et al., 2016]. This type of network became popular for computer vision problems after winning the largest contest in object recognition called ImageNet ILSVRC-2012 [Krizhevsky et al., 2012]. Since then, CNNs have become the standard form of neural network architecture for solving tasks where the raw data are images. The key aspects of CNN architecture are:

- parameter sharing: a feature detector (such as a vertical edge detector) that is useful in one part of the image is probably useful in another part of the image;
- sparsity of connections: in each layer, each output value depends only on a small number of inputs.

A layer being hidden or visible depends on its position in the network. It can also function in different manners. Types of layers in a CNN are convolutional, pooling, sub-, up- and down-sampling and fully connected. Image segmentation aims to extract segments from the background. There are primarily four types of techniques to do that:

- thresholding;
- boundary-based;
- region-based;
- hybrid techniques.

Thresholding assumes that image pixels correspond to either background or objects of interest that can be extracted by separating these pixels based on a threshold value. Secondly, we have the techniques that compare the pixel intensity to their local neighborhood and decide between discontinuity and similarity. Boundary-based methods, also called edge extraction methods, are the ones based on discontinuity. These types of methods assume that pixel intensity, color and texture should change abruptly between the regions. The region-based methods, on the other hand, are the ones based on pixel similarity and assume that neighboring pixels within the same region should have similar intensity and color values [Shih, 2010]. A combination of boundary- and region-based techniques also known as hybrid technique strives to achieve better segmentation.

3.2. Algorithm RefineNet

RefineNet is a generic multi-path refinement network that explicitly exploits all the information available along the down-sampling process to enable high-resolution prediction using long-range residual connections. It was introduced in 2016 and has achieved state-of-the-art results on seven public datasets [Lin et al., 2017a].

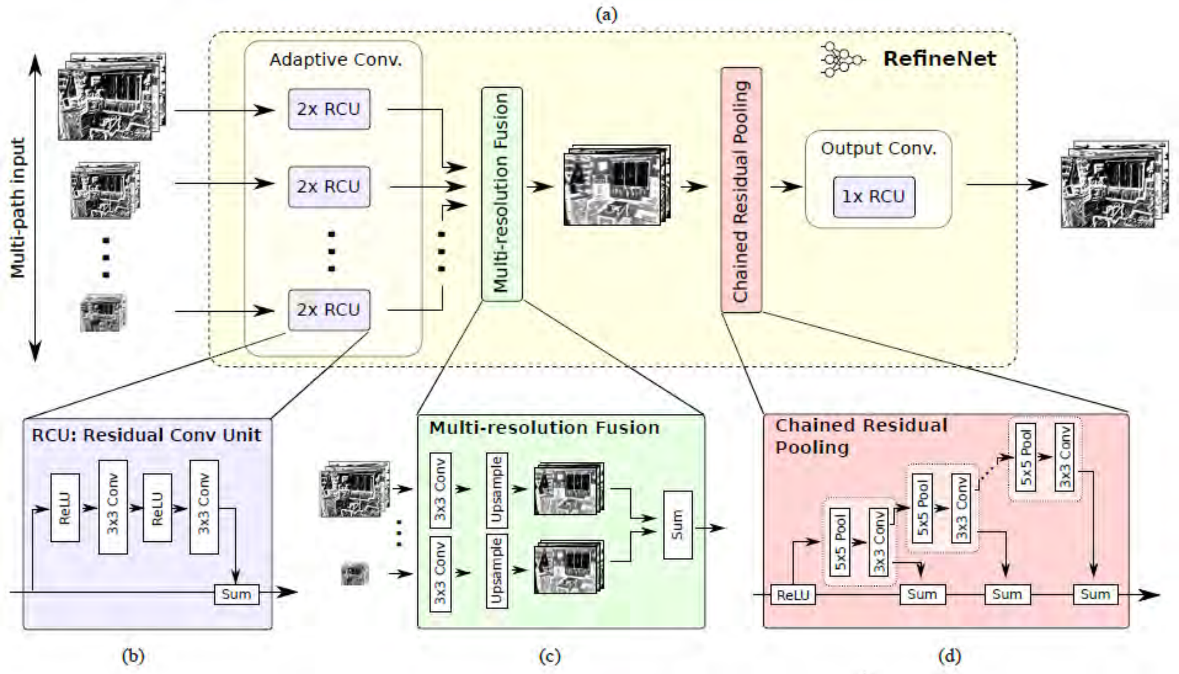


Figure 6.: Architecture flow: one RefineNet block

The proposed method is based on Fully Convolutional Network (FCN) [Long et al., 2015b]. It uses the very deep FCN architechture called Residual Net (ResNet) as a building block [He et al., 2016]. FCN based methods usually have the limitation of low-resolution prediction.

RefineNet consists of a number of specially designed components which are able to refine the coarse high-level semantic features by exploiting low-level visual features. It enables effective end-to-end training by exploiting the middle layer features.

Figure 6 shows one RefineNet block (a), and its components (b), (c) and (d) [Lin et al., 2017a]. Each block can be modified to accept any number of feature maps with any resolutions and depths. The Residual convolutional unit (RCU) shown in 6 (b) is an adaptive convolutional set that fine-tunes the pre-trained weights of ResNet. Each input-path is passed through two RCUs sequentially. The multi-resolution fusion block, shown in 6 (b), then fuses all paths into one high-resolution feature map. This block includes a convolutional, followed by an up-sampling layer and sums these up as a final step. The Chained residual pooling block shown in 6 (c) which aims to capture background context from a large image region. This pooling consist of a chain of pooling blocks. Each block has one max-pooling and one convolutional layer. The first block has one input path, while all other RefineNet blocks have two inputs. The RefineNet architecture is shown in Figure 7 [Lin et al., 2017a].

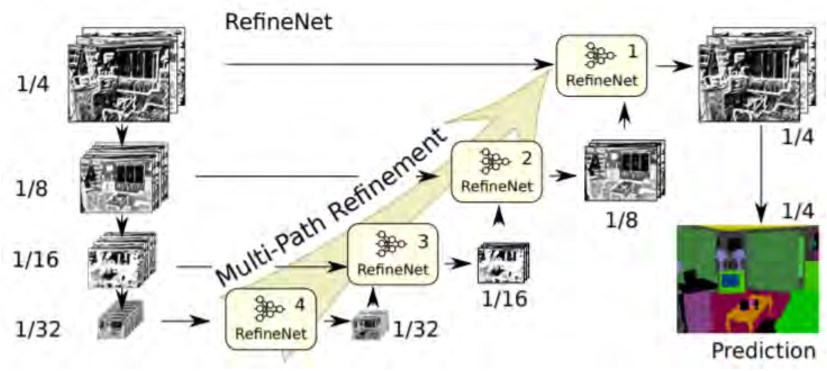


Figure 7.: Architecture flow: RefineNet

3.3. Algorithm Mask-RCNN

Mask R-CNN is a prominent member of the Regions with Convolutional Neural Network features (R-CNN) family [Girshick et al., 2014] that performs instance segmentation. Its brief history in chronological order is: CNN, R-CNN, Fast R-CNN and Faster R-CNN. In Subsection 3.1.2 we introduce the CNNs. Each of the other ancestors of Mask R-CNN are briefly explained as they are important in understanding how Mask R-CNN works.

3.3.1. History

R-CNN - Figure 8 shows the R-CNN object detection system overview [Girshick et al., 2014]. The architecture starts with an input image, extracts around 2k bottom-up region proposals, computes features for each proposal using a large CNN (with affine image warping) and finally classifies each region using a class-specific linear Support Vector Machine (SVM)s. This architecture achieves excellent bounding box mAP but has a notable disadvantage in terms of inference time. Object detection takes almost a minute on a Central Processing Unit (CPU) and 13 seconds on a GPU on a input image of size 227 by 227 pixels Red Green Blue, three-channel additive color model (RGB).

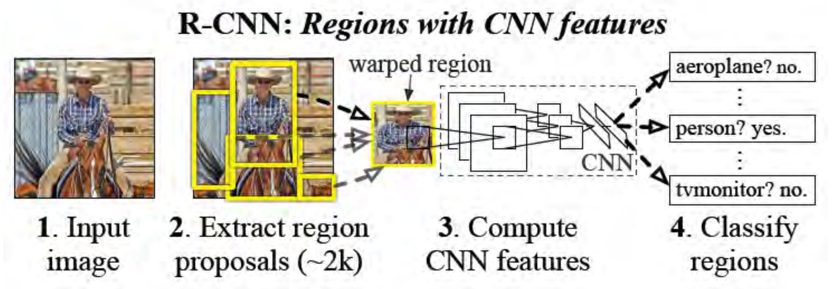


Figure 8.: Architecture flow: R-CNN

Faster R-CNN - The Fast R-CNN architecture improves the inference time and mAP of R-CNN by exposing region proposal computation as bottleneck [Girshick, 2015]. Additionally, the Faster R-CNN architechture introduces RPN that enables cost-free regional proposals. This architecture achieves state-of-the-art bounding box mAP with 300 proposals per image. The Faster R-CNN architecture consists of two modules: RPN that proposes regions and Fast R-CNN detector that uses the proposed regions.

The introduced RPN is similar to a FCN and can be trained for the task of generating detection proposals with a wide range of scales and aspect ratios. The anchor boxes serve as references at multiple scales and aspect ratios. Figure 9 shows the RPN architecture [Ren et al., 2015]. The proposals are generated by sliding a small network over the convolutional feature map output by the last shared convolutional layer. In the intermediate layer each sliding window is mapped to a feature that is for instance 256 dimensional. Finally, this feature is the input to two fully connected layers: a box-regression (reg) and box-classification (cls). The reg layer outputs the coordinates of v boxes, and the cls layer outputs scores that estimate probability of the presence of an object for each proposal. An anchor is the v proposals that are parameterized relative to v reference boxes [Ren et al., 2015].

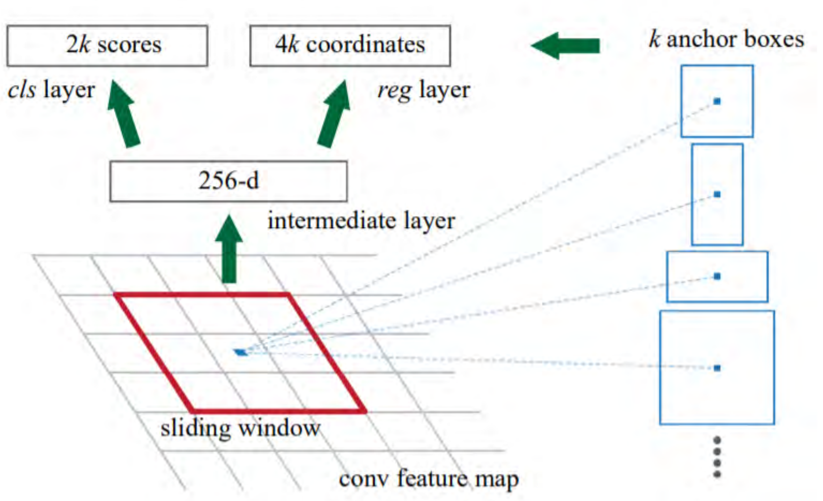


Figure 9.: Architecture flow: RPN

3.3.2. Mask R-CNN

Mask R-CNN is an extension of the object-detection architecture Faster R-CNN [Ren et al., 2015]. It adds a branch for predicting an object mask in parallel to the existing branch for bounding box recognition. Mask R-CNN outputs classification, bounding box and segmentation mask at pixel level for each detected object. Along with this enhanced performance, Mask R-CNN involves several hyperparameters which are to be tuned carefully based on the application. The official paper was published in 2017 and gained a lot of publicity by winning the COCO instance segmentation challenge [He et al., 2017]. The general flow of the Mask R-CNN architecture is shown in Figure 10 [He et al., 2017]. As shown, it consists of:

- Convolutional Network;
- RPN [Ren et al., 2015];
- RoIAlign;
- Two parallel output branches:
  - Object detection branch: bounding box and classification;
  - Segmentation branch: pixel-wise assignment of classes.

The compete Mask R-CNN consists of two main stages:

1. Backbone convolutional RPN that scans the image and generates proposals of interest;
2. Classification of the by step one generated proposals and generating bounding boxes and masks;

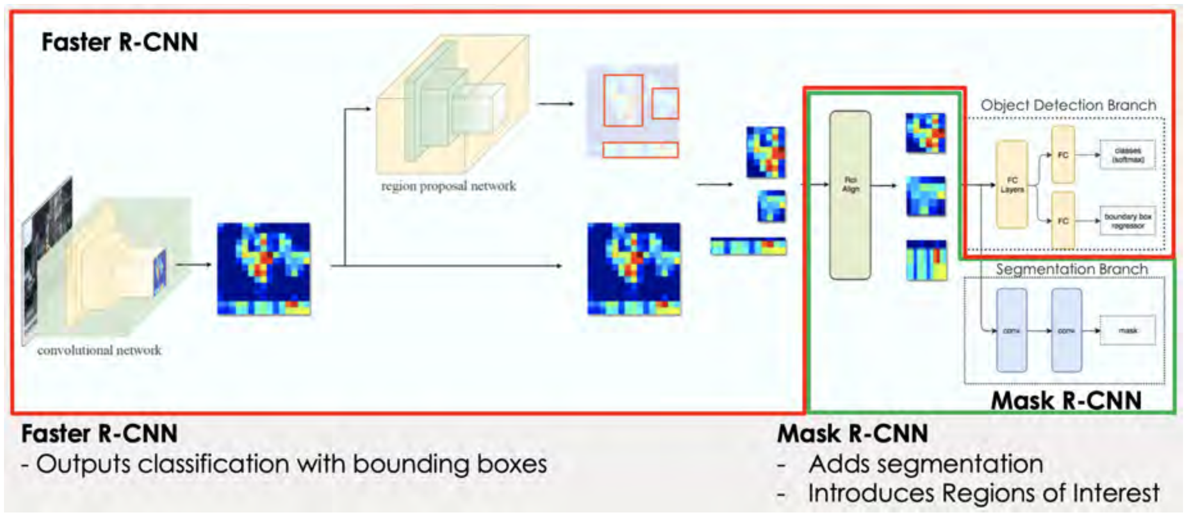


Figure 10.: Architecture flow: Mask R-CNN

The major contribution of Mask R-CNN what makes segmentation on top of Faster R-CNN possible is using RoIAlign instead of RoIPool for extracting a small feature map from each RoI. Next to this, the multi-task loss function is expanded by adding the mask loss to the total.

RoIAlign - The RoIPool performs coarse spatial quantization for feature extraction [He et al., 2015] and causes misalignment in the instance segmentation task. It is therefore replaced by the quantization-free layer called RoIAlign which uses bi-linear interpolation to compute the exact values of the input features [He et al., 2017].

Segmentation output branch - The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-wise manner [Long et al., 2015b]. This branch adds a small increase in computational overhead [He et al., 2017].

Multi-task loss function - The total multi-task loss on each sampled RoI equals

$$L = L_{cls} + L_{bbox} + L_{mask}. \tag{14}$$

The classification loss L_{cls} and bounding-box loss L_{bbox} are identical as those defined for Fast R-CNN [Girshick, 2015]. The mask loss L_{mask} is defined as the average binary cross-entropy loss. For an RoI associated with class k, L_{mask} is only defined on the k-th mask. The other mask outputs do not contribute to the L_{mask} loss [He et al., 2017].

4. Experimental Setup

In this chapter we discuss the preparatory steps that led to training the chosen algorithms and the training and inference details. These steps are:

- getting familiar with the dataset that is documented in Section 4.1,
- pre-processing the data to ensure optimal learning conditions handled in Section 4.2,
- the training details described in Section 4.3,
- the inference details described in Section 4.4.

4.1. Problem instances

The open source data that was found online is the iMaterialist dataset. It is available on Kaggle as a 19GB file already split in a train and test set and is accompanied by a label description file and an annotation file for the train set. The competition based on the dataset is part of the Fine-Grained Visual Categorization FGVC6 workshop at the Computer Vision and Pattern Recognition Conference CVPR 2019 [iMaterialist Fashion Competition group, 2019]. We chose this dataset for its size, the fashion classes and the pixel-wise annotation. Making a dataset of this size and quality that includes this variety of classes and annotating each segment is extremely costly and time consuming. Thus we were extremely eager to train our models on it.

The iMaterialist dataset consists of 45.195 annotated RGB images. The variation in displayed apparel, number of people in the picture, poses and setting make it a suited dataset for training. Each image has a unique ID and each apparel class has a different name. The images vary in size, with a minimum of 257 by 636 and maximum of 10.717 by 6.824 image pixels. In general, the images are considered of good quality and present clear objects (see some examples below in Figure 11).

There are a total of 46 apparel categories (27 main items and 19 garment parts, closures and decorations). An examples of main category is '0: shirt' or '23: shoe'. These categories are divided into 12 supercategories, see Table 4.

The 45.195 images consist of 331.213 segments. One segment is defined as all pixels belonging

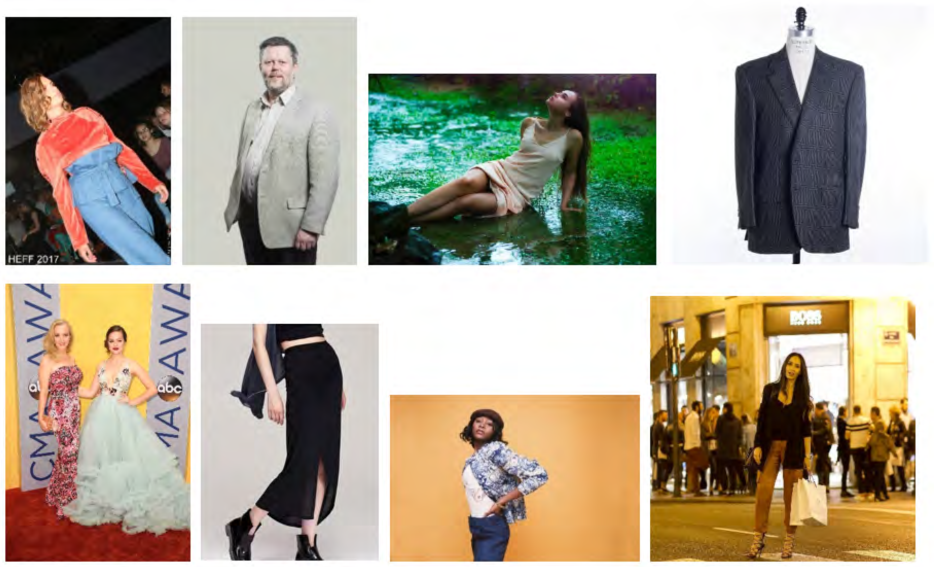


Figure 11.: Example images of the iMaterialist dataset

Table 4.: Label supercategories and categories

	Supercategory	**Categories**
1	wholebody	dress, cape, coat, jumpsuit
2	upperbody	shirt, blouse, top, t-shirt, sweatshirt, sweater, cardigan, jacket, vest
3	lowerbody	pants, shorts, skirt
4	arms and hands	watch, glove
5	waist	belt
6	head	glasses, hat, headband, head covering, hair accessory
7	neck	tie
8	legs and feet	leg warmer, tights, stockings, sock, shoe
9	others	bag, wallet, scarf, umbrella
10	garment parts	hood, collar, lapel, epaulette, sleeve, pocket, neckline
11	closures	buckle, zipper
12	decorations	applique, bead, bow, flower, fringe, ribbon, rivet, ruffle, sequin, tassel

to one fashion item. The occurrence of each category in the train dataset is shown in Figure 12.

There is a total of 92 related apparel fine-grained attributes. Examples of attributes are '0: above the hip' or '40: tight (fit)'. These attributes are divided into 7 supercategories: length; opening type; silhouette; textile finish, manufacturing techniques; textile pattern; animal; waistline. Seventeen of the categories have attributes, but only 3.47% of the segments actually have these fine-grained labels. In Section 4.2 we explain how we pre-processed the data and the reasoning behind it.

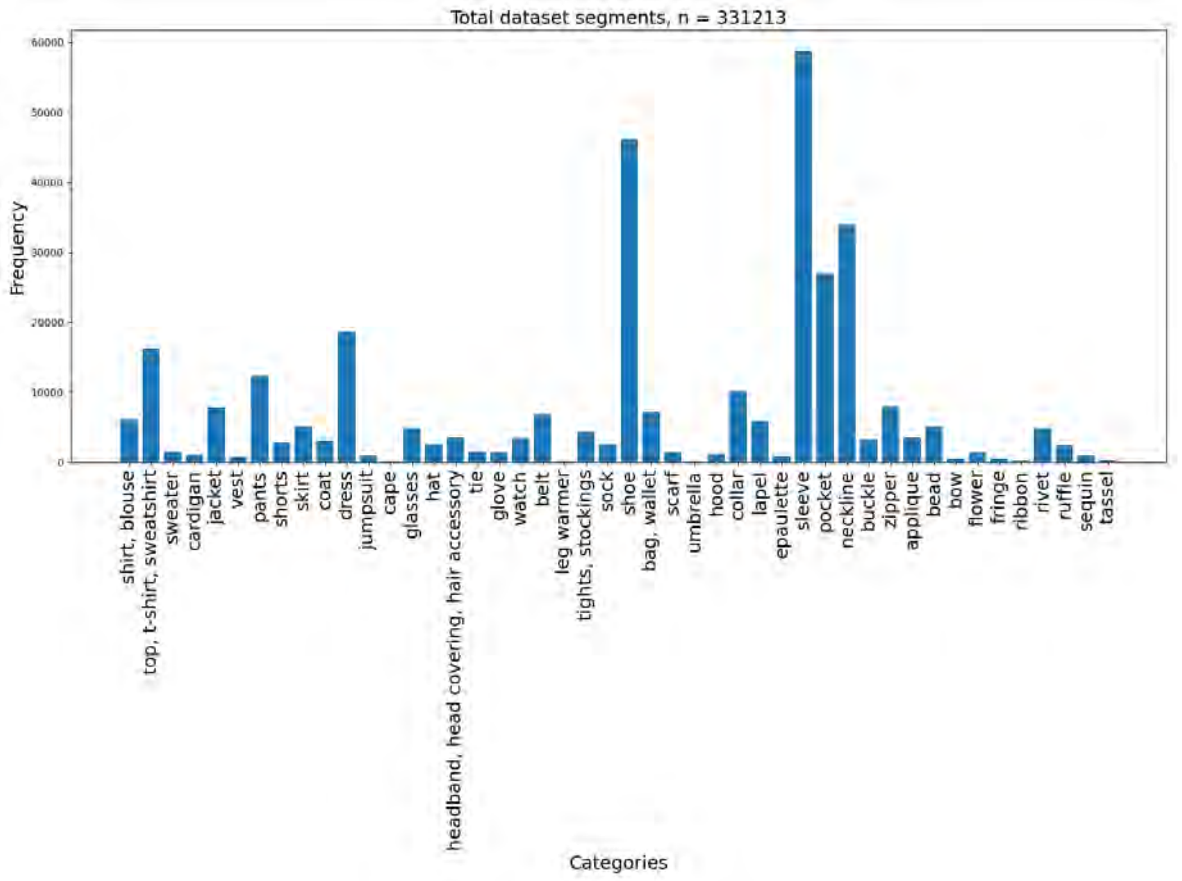


Figure 12.: Total number of segments in train dataset per category

4.2. Data pre-processing

Most of the data pre-processing steps seem standard to any machine learning task. However, even data cleaning and normalizing comes with a twist in the field of computer vision. There are also pre-processing steps as image augmenting and resizing that are specific for the field. There are five steps between the raw data images and the final data that was fed into the algorithms for training:

1. Data label reduction: sub-setting the images and their annotations to only the ones containing the chosen set of labels. The base set on which the initial experiments are performed consisting of five classes is 'shoe', 'dress', 'top, t-shirt, sweatshirt', 'pants' and 'jacket'. The extended list on which the final model is trained consisting of eleven classes is 'shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'jacket', 'pants', 'skirt', 'dress', 'tie', 'shoe', 'bag, wallet' and 'umbrella'.
2. Data resizing: bringing all of the images and their annotated pixels to the uniform size of 512 by 512 keeping the original ratio and padding it with zeros;

3. Data cleaning: manually correcting for any mislabeled segment when found;
4. Data normalizing: rescaling each image pixel by deducting the channel mean of the total set (123.7, 116.8, 103.9) from its respective RGB value;
5. Data augmenting: transforming a randomly chosen portion of the images by performing a combination of the techniques horizontal flip, blur, noise addition and brightness adjustment to each image.

4.2.1. Data label reduction

The set of images that is used is chosen based on the occurrence of an item in the dataset and the usability of the class to this project. The label set is limited to:

- no class attributes such as 'fit' and 'length';
- complete clothing item classes meaning no garment parts, closures and decorations such as 'sleeve', 'zipper' and 'ruffle';
- base and final set including respectively a subset of five and eleven selected classes.

The chosen base and final set of classes are shown in Table 5. Appendix C shows an example of original images and each annotated fashion item belonging to the final set.

Table 5.: Base and final set of classes with their annotations

Base set	**Full set**	**Annotation**
'top, t-shirt, sweatshirt'	'shirt, blouse'	0
'jacket'	'top, t-shirt, sweatshirt'	1
'pants'	'sweater'	2
'dress'	'jacket'	3
'shoe'	'pants'	4
-	'skirt'	5
-	'dress'	6
-	'tie'	7
-	'shoe'	8
-	'bag, wallet'	9
-	'umbrella'	10

Exclude class attributes

Many of the categories have attributes. That means, that for instance the category 'dress' can have attributes like 'knee (length)', 'asymmetrical', 'a-line', 'lining' etc. These attributes are excluded as they make the problem far more complex, but have an insignificant contribution to the quality of the results as the client is not interested in it and only about 3% of the segments have attributes. This does not change the total amount of images and segments.

Exclude garment parts, closures and decorations

In the list of main clothing classes, not only the complete items, but also the parts that they are made of are included. Those are: ’collar’, ’lapel’, ’epaulette’, ’sleeve’, ’pocket’, ’neckline’, ’buckle’, ’zipper’, ’applique’, ’bead’, ’bow’, ’flower’, ’fringe’, ’ribbon’, ’rivet’, ’ruffle’, ’sequin’ and ’tassel’. As we are not interested in these garment parts we delete their segments from the annotations file. This leaves us with one image less ($i = 45.194$) and roughly half of the segments ($s = 163.371$).

Choose a shortlist of five classes

We are left with twenty eight complete items to consider. The top five in occurrence are ’shoe’, ’dress’, ’top, t-shirt, sweatshirt’, ’pants’ and ’jacket’. These five make a good base set in terms of the supercategories. ’shoe’ belongs to legs and feet; ’dress’ to wholebody; ’top, t-shirt, sweatshirt’ and ’jacket’ to upperbody and ’pants’ to lowerbody. This base set makes op for 93% ($i = 42.031$) of the images and 62% of the segments without garment parts ($s = 101.284$).

Choose a final extended list

Once it has been established that the model is capable of learning the shortlist we want to train it with a more challenging and complete set of items. The final extended list is influenced by:

- base subset of five most frequently occurring classes;
- class annotation of the client dataset;
- pre-learned overlapping classes from the COCO weights.

This list is first and foremost based on the results of the learning with the shortlist. We experienced the model mistaking bags for shoes as it had no other item to define it with and, for instance skirts being mistaken for dresses or pants. Secondly, based on the classes contained in the annotated client images. These classes are: ’shirt, blouse’, ’top, t-shirt, sweatshirt’, ’sweater’, ’jacket’, ’pants’, ’skirt’, ’dress’, ’shoe’. This is explained in more detail in Chapter 5. The third influence to this sub-set is that we are using COCO pre-trained weights and three of the classes are the same or similar and can be used as is. Those are ’tie’, ’bag, wallet’ and ’umbrella’. This is explained in more detail in Subsection 4.3.1.

The three factors make for the final extended set consisting of the eleven classes represented in Table 5. This list consists of classes of six out of the nine relevant supercategories. ’shoe’ belongs to legs and feet; ’dress’ to wholebody; ’top, t-shirt, sweatshirt’, ’shirt, blouse’, ’jacket’ and ’sweater’ to upperbody; ’pants’ and ’skirt’ to lowerbody; ’bag, wallet’ and ’umbrella’ to others; ’tie’ to neck. This extended set of eleven classes makes up for 98% of the images ($i = 44.234$) and 75% of the segments without garment parts ($s = 122.704$).

4.2.2. Data resizing

Even though it is nice to have images of high quality, each pixel adds extra calculations to the training and inference of the algorithm. Thus a reduction of the image size is expected to contribute to faster inference [He et al., 2017]. Furthermore, establishing a base size is common practice in computer vision in order to support batch training.
The images in the training data vary in size. The height of the images ranges between 296 and 8.688 pixels. The width ranges between 257 and 10.717 pixels. Most images have larger height than width. We must consider this when resizing as the ratio (height / width) varies per picture.
To support multiple images per batch, images are resized to one uniform size. The image size, inference time and quality of the prediction are related. The bigger the image, the slower the inference but the better the prediction. As there is the bounding condition of a maximum of two seconds for the inference of one image, we looked for the highest possible size that achieves this goal. We considered 1024 by 1024, 512 by 512, 256 by 256 and 128 by 128 pixels and found that an image size of 512 by 512 pixels is the largest size to fit the requirements. This is relieving as it is a relatively large size and thus preserves most of the information of each image. In order to also preserve the aspect ratios of the images instead of stretching it to one size fits all, we chose to resize each image to a square. First, it is resized to fit into a 512 by 512 pixels window and then padded with zeros to fill the gaps if the original shape was not a square.

4.2.3. Data cleaning

Data generating process is manual and thus it is possible that mislabeling can occur even though there is a strict protocol. However, it is hard to look for mislabeled data, because that would mean going through more than 40.000 images and their labels. Even so we did stumble across one example that was clear while experimenting. Figure 13 shows an image labeled 'legwarmer' while the fashion item is 'shoe'. The annotation of this example is corrected in the annotation file. There might be more mislabeled images, but those have not been discovered by us and we did not find a report about it online.

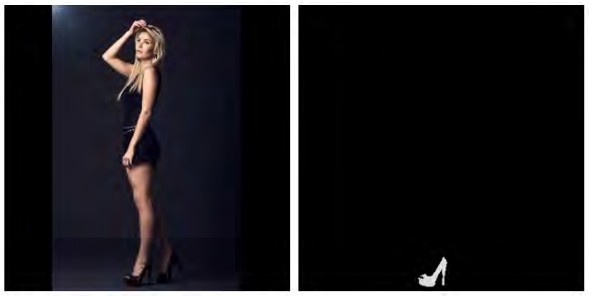


Figure 13.: Image (left) and mislabeled segment (right)

4.2.4. Data normalization

The data to be normalized in computer vision are the RGB values of each pixel of the images. These are naturally scaled and range between 0 and 255 for each channel. It is common practice to normalize these values to either:

- zero mean called pixel centering;
- range [0,1] called pixel normalization.

This helps the algorithm to converge faster and prevents vanishing and exploding gradients. Furthermore, when the data is not normalized, the input with highest means will contribute the most to the training process and normalizing is a way of preventing that. We found that centering is the most popular technique. Specifically, subtracting the mean over the whole training dataset is being used in many of the most successful and broadly used algorithms. We did not calculate the iMaterialist channel mean but used the COCO dataset mean of 123.7, 116.8, 103.9. We also experimented with dividing the pixels by 255 and pixel normalization [Lecun et al., 1998]. The most notable difference in performance and convergence time is measured between having no normalization or implementing one of the methods rather than between the methods itself.

4.2.5. Data augmenting

Data augmenting is a versatile method for making the dataset at hand more diverse and is also considered a regularization method. Due to its popularity there are many augmentation techniques. Some examples are rotation, flip, shift, cutout, shear, invert and mix-up. There are so many techniques that also reinforcement learning can be used in order to find the best performing ones like in the paper AutoAugment: Learning Augmentation Policies from Data [Cubuk et al., 2018].
However, we handpicked the methods to use in this case. We apply image augmentation transformations only on our training images. The client of the CP&R project has provided a sample of images taken in a shopping mall to test the final model on. These sample of images are of average quality, a bit blurry and the lighting is poor. On the other hand, the iMaterialist dataset consists of professional images of prime quality and great detail. Thus augmenting images in terms of this project is meant to fill the gap between the iMaterialist dataset and the images that the model is going to be used on, while keeping in mind that we carefully selected a set of four techniques from the Imgaug library [Jung et al., 2020]:

- Gaussian Blur uses a Gaussian kernel with a random standard deviation sampled uniformly (per image) from a chosen interval. The interval we chose is 0 to 1 being no blur to some blur, respectively.

- Additive Gaussian Noise adds Gaussian noise to an image, sampled once per pixel from a normal distribution $N(0, s)$, where s is sampled per image from a chosen interval. We chose s between 0 and $0.05 * 255$.
- Add random values in the chosen range of -40 to 40 to images. The sampled value is applied to all pixels in the image. This augmentation effects the brightness of the image.
- Horizontal flip is a very common augmentation strategy that turns the image left or right.

Figure 14 demonstrates the chosen augmentation strategies [Jung et al., 2020]. We do not want to alter the dataset too drastically and thus do not use all of the augmentations on each image. The application scheme is as follows: approximately half of the images are being flipped; we add a value between -40 and 40 to all pixels of approximately half of the images; we apply either Gaussian Blur or Additive Gaussian Noise to approximately half of the images. Putting this scheme in practice means that we can have an image that undergoes zero to three augmentations those being flip, brightness adjustment and either blur or noise.

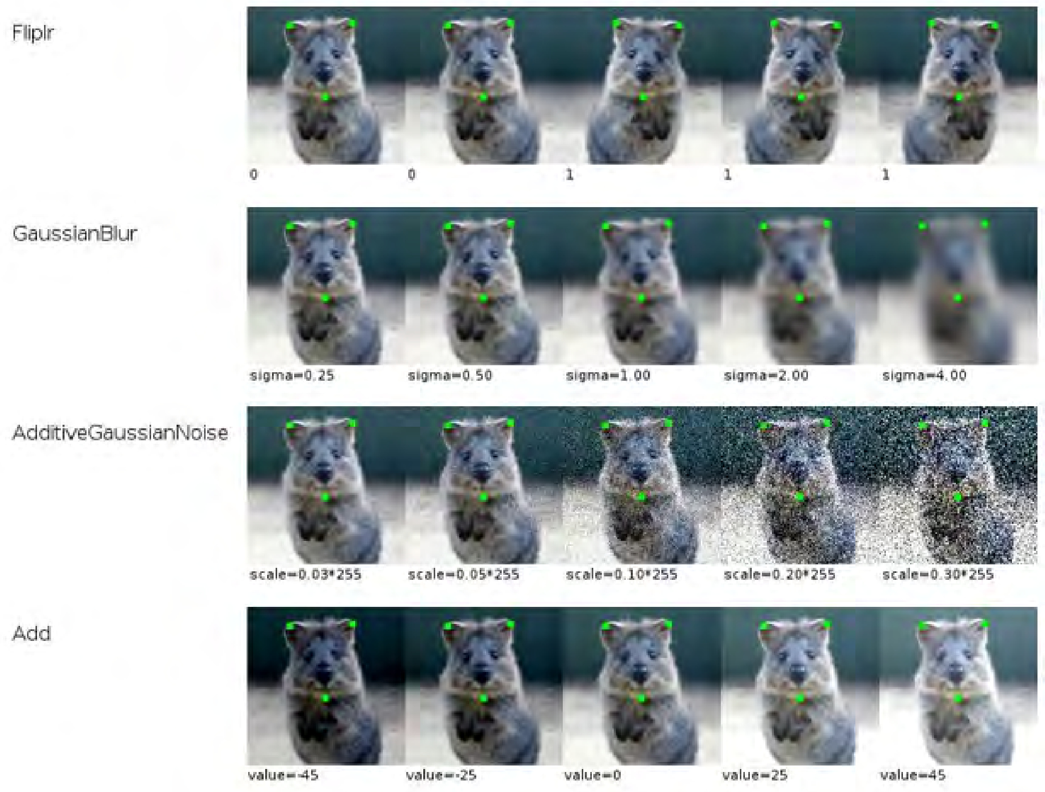


Figure 14.: Augmentation strategies applied

4.3. Training details

The training is performed on the pre-processed data, making use of the model architectures as specified in Chapter 3. An essential part of training the Mask R-CNN model is making use of Transfer learning that is explained in depth in Subsection 4.3.1. In Subsection 4.3.2 we discuss the training scheme and parameters of Mask R-CNN.

Regularization - In order for the model to be able to apply the learned patterns from the training data to the validation and test data we need to include regularization methods that make sure this happens. One of this methods is data augmenting what we already discussed in 4.2.5. The other used method is weight decay.

Hardware training - All training is performed remotely using external GPU computing power on Amazon Web Services (AWS) to speed up the learning process. We use NVIDIA Tesla M60 GPUs with 8 GiB of GPU memory and 4 vCPUs. Each mini-batch has 2 images per GPU.

4.3.1. Transfer Learning

Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones. In this definition, transfer learning aims to extract the knowledge from one or more source tasks and applies the knowledge to a target task. It allows for the source and target data to have different distribution, labeling function and even consist of different classes [van Opbroek et al., 2015]. Transfer learning can be categorized in three sub-settings:

- inductive transfer learning, where the source and target domains are the same, while the tasks are different but related;
- transductive transfer learning, where the tasks are the same, while the domains are different but related;
- unsupervised transfer learning, where the domains are the same, while the tasks are different but related [Pan and Yang, 2010].

It helps the algorithm by piking up pre-trained weights from a different problem instead of training from its initial weights. This helps overcome the model's bias towards training data and domain, making learning faster and preventing overfitting. We have experimented with transferring knowledge from ImageNet or COCO dataset. Initially, we compared the classes. iMaterialist and COCO have three classes in common next to the fact that due to COCO weights the model already could recognize people in the image. This is one of the underlying problems for segmenting fashion items as people are wearing them. Secondly, training with

COCO weights outperformed training with ImageNet significantly comparing the validation loss of the tenth epoch using the same setting. In this case the source data and target data have different labeling functions, as well as different features and prior distributions, but the tasks are both image segmentation.

In order to confirm a positive transfer with COCO pre-trained weights we inferred it on a small set of iMaterialist data, see Figure 16 for a small selection of images showing a positive transfer of the COCO classes ’person’, ’tie’ and ’umbrella’.

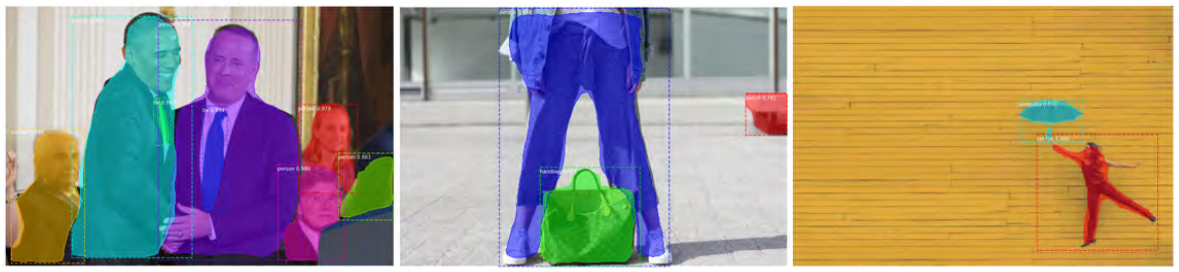


Figure 15.: Transfer Learning: utilizing COCO pre-trained weights overlapping classes

See Figure 16 for a small selection of images where improvements need to be made. From left to right it shows that the blurry people in the background are classified as ’backpack’, the image in the center containing a top is labeled ’vase’ and the dress ’bird’.

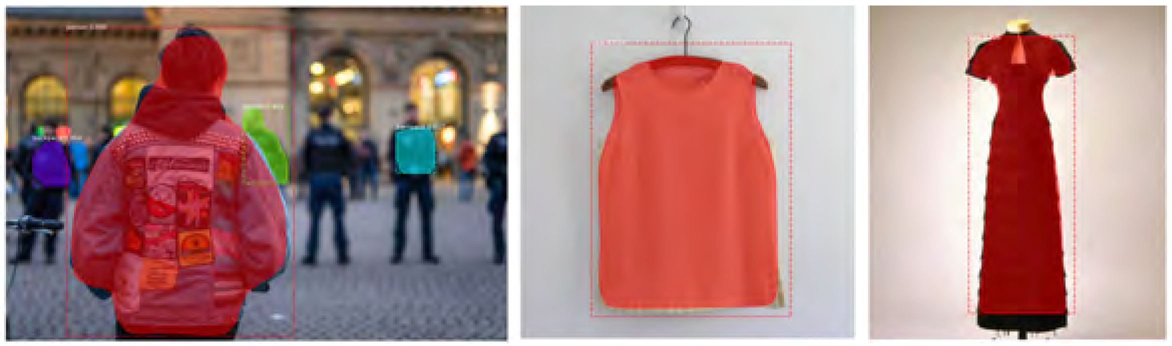


Figure 16.: Transfer Learning: utilizing COCO pre-trained weights

4.3.2. Training parameters Mask R-CNN

As the positive transfer is confirmed, the training on the iMaterialist dataset is executed using pre-trained COCO weights. We utilized an open-source implementation of the Mask R-CNN model provided by Matterport [Abdulla, 2017]. The training is done in multiple runs in which we experimented with different hyperparameters and learning schemes in order to achieve the lowest validation loss. As mentioned earlier we train first on the five base classes and name those experiments Base1 to 4, followed by experiments on the final set of eleven classes, named Full1 to 4.

We differentiate the training over the convolutional *backbone* architecture used for feature extraction over an entire image and the network *head* for bounding-box recognition (classification and regression) and mask prediction that is applied separately to each RoI. The general strategy for each run is to train the head layers of the architecture without any data augmentation with a constant learning rate for some epochs and then to train the model end-to-end by training all layers with data augmentation decaying the learning rate. The algorithm has 48 hyperparameters, see Appendix B. We focused on a couple and left the other to their default settings. The ones we have focused on are:

1. Backbone architectures available in the Matterport implementation are ResNet50 and ResNet101 on top of which we add a Feature Pyramid Network [Lin et al., 2017b]. As shown in the official Mask R-CNN paper, the deeper the ResNet the more accurate [He et al., 2017]. However, there is a trade-off between training time and accuracy, but given that we aim for the highest accuracy and it is the preferred backbone in the official Mask R-CNN paper we decide to use ResNet101-FPN.
2. The default configuration uses image shape of (1024, 1024, 3) being the height, width and number of channels, however, we saw that this image size is too heavy and does not suffice the inference time limit. We trained with an image shape of (512, 512, 3).
3. The learning rate in the official paper α =0.02 is decreased by 10 at the 120k iteration. This, however, seems too high to start with and is adjusted to α =0.0005 for the first epochs where we train the heads only. The end-to-end training starts with a learning rate of 0.0001 and decays to 0.00001.
4. Length of square anchor side in pixels default of (32, 64, 128, 256, 512) is adjusted to (16, 32, 64, 128, 256). The default setting anchors are very big proportionally for the image size chosen by us. We do not have any fashion items that occupy the full image so the maximum anchor for us is of size 256 and as we do have some fine detailed items we added the smallest anchor of size 16.
5. Steps per epoch is set to 5000 for most runs while the parameters are being tuned as training is very expensive. The final run is performed with 15000 steps per epoch.

In total, we train all runs using SGD with momentum of 0.9. We use a batch size of 2 and perform the training remotely using external GPU computing power to speed up the learning process. Gradients are clipped to 5.0 and weights are decayed by 0.0001 each epoch. We conducted additional experiments with other learning rate schedules, but this scheme leads to the results that showed the best results and are reported in Chapter 5. See Table 6 for the run specific settings that were used to train the models of which we also show the results in Chapter 5.

Table 6.: Train specifics

Run	Best epoch	Duration	Im_size	Augment	Epochs	Steps	Pre-trained
Base1	20	8.5 h	512	Fliplr	20	1.000	COCO
Base2	30	9 h	512	All four	30	1.000	COCO
Base3	4	30 h	512	All four	4	18.078	COCO
Base4	7	4 h	256	Fliplr	8	1.000	ImageNet
Full1	10	45 h	512	Scheme 4.2.5	30	5.000	COCO
Full2	8	12 h	512	Scheme 4.2.5	10	5.000	COCO
Full3	10	16.5 h	512	Scheme 4.2.5	10	5.000	COCO
Full4	28	18 h	512	Scheme 4.2.5	30	5.000	COCO

4.4. Inference details

Hardware inference - A requirement of the CP&R project is that the final model is to be used in real-life environment on a computer comparable to the one Bright Cape employees work on. In specific it is a computer with an Intel Core i7-7820HQ Processor CPU with 4 cores and 8 threads [DELL, nd] and n NVIDIA GeForce 940MX GPU [NotebookCheck, nda]. This means that this computer has a relatively powerful CPU and a weaker GPU, which directs us to focus on CPU processing.

This contrasts with the most used setups on which most models are inferred. Mask R-CNN for instance is inferred using a NVIDIA Tesla M40 GPU [NotebookCheck, ndb] which is a professional graphics card [He et al., 2017]. This is why even though the authors report their inference time ($\tau = 0.2$ seconds) it is not comparable to what we can achieve using our specifications.

4.4.1. Inference parameters Mask R-CNN

We found in the literature that in order to speed up the inference we can change the maximum number of detection instances. The default is 100, but even if a person is wearing all of the fashion items and counting both shoes as individual, the maximum of segmented instances is twelve. Given that there are fashion items that mutually exclude eachother, like blouse and top, and skirt and pants we are confident with setting this maximum to ten. This reduction is implemented during inference. This reduces the speed from 2.1 to an average of under 1.5 seconds for an image of size 512 by 512 pixels. The final results, shown in Chapter 5.4 are inferred using the reduced maximum of detection instances.

5. Experimental Results

The algorithms have been described and compared in terms of their complexity in Chapter 3. In this chapter we show the evaluation of the performance in terms of capability to classify, detect and segment the present objects in an image and the time it takes to do so. The results have been derived from two test sets M_{test} and C. These are described in Section 5.1. The results shown in this chapter concern precision in predicted items measured in EMR, OA and JSS for the task of multi-label image classification, and F1-score and mAP for the task of segmentation. The inference time for one test image in measured in seconds. In order to quantify the metrics for the different tasks, speed and datasets as one score per model, in Section 5.2 we present the novel metrics:

- Classification score,
- Segmentation score,
- Speed score,
- Trade-Off score (TO),
- Total Trade-Off score (TTO).

The model with the highest TTO is defined as the best. The results derived using the RefineNet and Mask R-CNN algorithm are shown in Section 5.3 and 5.4, respectively. The final and best model is discussed in Section 5.5.

5.1. Test instances

Some parts are omitted due to confidentiality of the client data

M_{test} is a portion of the pixel-wise annotated iMaterialist train set that was held back during training. It contains 1.000 images and 2.777 segments. The original released iMaterialist test set is not used as it is not annotated and we cannot quantify the performance on it.

C contains images that have been taken on location where the final model is to be used in practice. It contains 460 images and 1968 segments. It was provided by the client of the CP&R project. This set provides a more solid idea of the performance of the model as the images match the lighting, poses and setting of the conditions of the actual use for the model. However, it contains per image annotation and not per pixel.

The images are labeled by hand using annotation software. The labeling protocol differs from the iMaterialist one so some adjustments have been made. The original annotations had the values: 'blouses', 'shirt', 'sweater', 'jacket', 'pant', 'skirt', 'dress', 'jean', 'leggings', 'shoes'. After some inspections of the images we chose to make the following adjustments:

1. the label 'blouses' can be mistaken for the 'shirt, blouse' label from the iMaterialist dataset, but the objects look more like tops and t-shirts, so we decided to keep it as a separate class and rename it as the iMaterialist 'top, t-shirt, sweatshirt';
2. the label 'shirt' annotates mostly clothing items with a collar and/or buttons and can be renamed to 'shirt, blouse';
3. 'jean' is pants with extra information about the textile being denim, but as we have decided not to use attributes, the labels 'pant' and 'jean' are merged to the 'pants' category of the iMaterialist dataset;
4. because the label 'leggings' occurs only once in the whole annotation set we could determine that it is actually pants by looking at the image and comparing it to other annotations. It is a skinny jeans that is worn in the next image, but there it is labeled as pants, this is the reason why we convert the 'leggings' label to 'pants';
5. we relabel 'shoes' to two times 'shoe'. The difference in annotation here is that the project counts a pair and in the iMaterialist dataset each shoe is an object with a separate mask.

Table 7 shows the original and new annotations and occurrences.

Table 7.: Annotations of the CP&R test images

	Original annotation	**Original count**	**iMaterialist label**	**Action**
1	blouses	199	top, t-shirt, sweatshirt	converted
2	shirt	93	shirt, blouse	converted
3	sweater	92	sweater	no change
4	jacket	275	jacket	no change
5	pant	170	pants	converted
6	skirt	93	skirt	no change
7	dress	61	dress	no change
8	jean	120	pants	converted
9	leggings	1	pants	converted
10	shoes	423	shoe, shoe	converted

The data label cleaning step leaves eight classes distributed over a total of 1.536 segments and 459 test images. The total occurrence of each class can be seen in Figure 17.

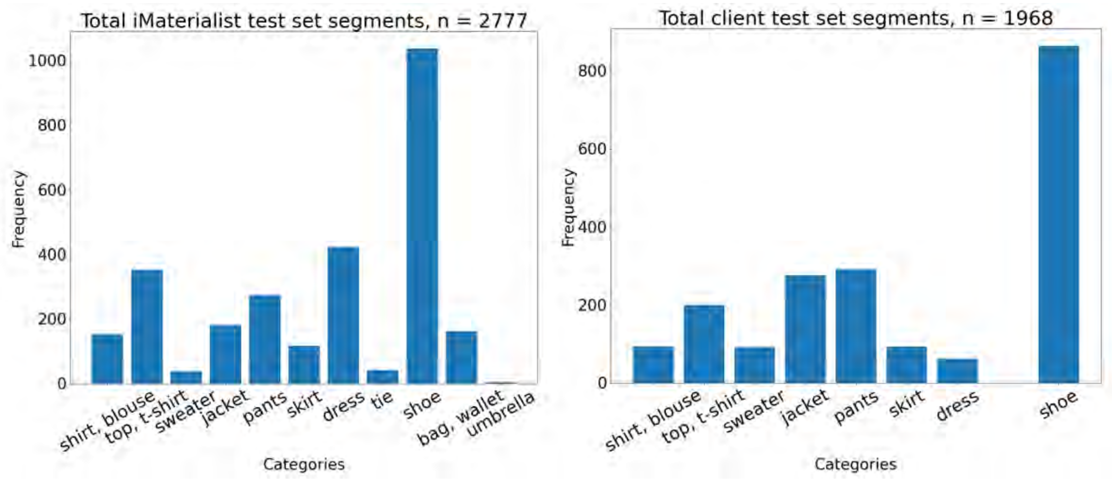


Figure 17.: Distribution test dataset segments after relabeling

We resize all test images before inference as discussed in Subsection 4.2.2. Due to the difference in annotation of the two test sets the models can be evaluated on both M_{test} and C for their performance on the multi-label image classification task, but only on M_{test} on their performance on the task of image segmentation.

5.2. Definition of best model

In order to compare the performance of multiple models on the two available datasets we define a novel metric with which we can quantify the trade-off between multi-label classification, segmentation and inference time. First, we combine multi-label classification metrics EMR (1), OA (2) and JSS (3) into one weighted multi-label image classification. We assign EMR the highest weight as in the best case scenario all images will be exactly classified

$$Classification\ score = \frac{3 \cdot EMR + OA + 2 \cdot JSS}{6}. \tag{15}$$

To illustrate the Classification score we calculate it for the example shown in Table 2:

$$\frac{(3 * 0\% + 2 * 75\% + 80\%)}{6} = 38.3\%. \tag{16}$$

This shows that this metric inherits mostly the harshness of the EMR. Even though the metric has a range of [0,1] we consider a Classification score above 0.4 good classification and above 0.5 very good classification.

To evaluate the pixel-wise segmentation we take the average of the mAP (10) and F1-score (8):

$$Segmentation\ score = \frac{mAP + F1_{score}}{2}. \tag{17}$$

To access the inference time we divide the average inference time T$_{inf}$, defined in Equation (11) by 10 to lessen the magnitude, as shown in Equation (18):

$$Speed\ score = \frac{T_{inf}}{10}. \tag{18}$$

TO is the combination of the three components. We sum the Classification and Segmentation score and penalize with the Speed score embodying the trade-off between them. It can be computed for M_{test} using

$$TO = Classification\ score + Segmentation\ score - Speed\ score \tag{19}$$

equal to

$$Equation(15) + Equation(17) - Equation(18). \tag{20}$$

We value the Classification score of C higher than of M_{test} because C is more representative of the setting and this score is its only contribution to the total performance we weight it higher. This reasoning brings us to the final metric:

$$TTO = Classification\ score(C) * 1.5 + TO(M_{test}), \tag{21}$$

equal to

$$TTO = Equation(15) * 1.5 + Equation(19), \tag{22}$$

The best model is defined as the one with highest TTO. The added value of this metric is its capability to discriminate among model results by combining all the single metric results for both datasets and taking into account the importance of each task performance on each of the datasets.

5.3. Results Algorithm RefineNet

Experiments with the RefineNet architecture failed to produce usable fashion segmentation. We ran 22 experiments using different:

- sub-sets with variation of either one, two or three classes. The initial experiments include only the relatively easy to learn and well represented in the dataset classes 'pants', 'shoe' and 'jacket';
- pre-processing as explained in Section 4.2 being:

  - resizing style: cropped and padded;
  - image size: 384 by 384, 256 by 256 and 128 by 128 pixels, RGB;
  - normalizing: pixel centering and pixel normalization;
- training parameters being:
  - loss functions with and without class weights: dice loss, categorical crossentropy, weighted crossentropy and the custom ignore_unknown_crossentropy;
  - optimization functions: SGD and Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014];
  - learning rates, number of epochs and steps per epoch.

With a cumulative training time of more than 200 hours, the segmentation results of 18 runs are all background, as shown in the left image of Figure 18. Three of the runs predict all only one class, meaning all 'pants' or all 'jacket'. In the right image we see that the particular model has segmented large areas of the image as fashion items, but in reality those seem rather random as the light-blue is supposed to be the class 'pants' and the slightly darker blue 'jacket'. We discuss these result in more detail in Subsection 6.2.1.

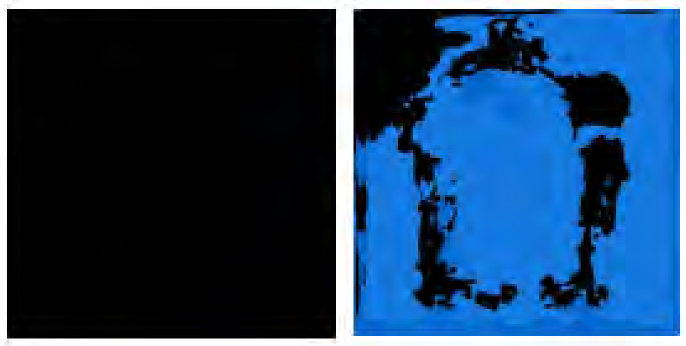


Figure 18.: Results RefineNet

5.4. Results Algorithm Mask R-CNN

We run a total of 20 experiments. The total multi-task loss on each sampled RoI is as shown in Equation (14). For each experiment we log the following values:

- Train and validation loss L per epoch;
- Train and validation L_{cls} per epoch;
- Train and validation L_{bbox} per epoch;
- Train and validation L_{mask} per epoch;

- EMR on the Client dataset and iMaterialist testset;
- OA on the Client dataset and iMaterialist testset;
- JSS on the Client dataset and iMaterialist testset;
- mAP@50 on the iMaterialist testset;
- F1-score on the iMaterialist testset;
- T_{inf} on the iMaterialist testset;

The factors determining how well an algorithm will perform are its ability to minimize the training error and the gap between training and test error. These two factors correspond to the two central challenges in machine learning underfitting and overfitting. In the case of underfitting, the model is not able to obtain a sufficiently low error value on the training set. In the case of overfitting, the training error is low but the test error is significantly larger, making the train-test gap too large [Goodfellow et al., 2016].
Plots of the train and validation L, L_{cls} and L_{mask} of the eight experiments are shown in Appendix D. We observe decreasing train and validation loss for all eight experiments with signs of convergence for experiment Base1, Full2, Full3 and Full4. During training we test after each epoch on a validation set. The gap between the train and validation loss shows decreasing trend. This indicates that the model is capable of learning the patterns in the data.

The metrics evaluated on the test set after training are shown below. The run name indicates whether the base set of five labels or the full set of eleven is used for the corresponding run. The base and full set are defined in Section 4.2 in Table 5.

Evaluation on Client dataset
Because of the missing pixel annotations of the 460 client images, only the multi-label image classification metrics are shown for that dataset. Table 8 presents the evaluation metrics. As discussed in Section 2.3 we use EMR, OA and JSS defined in respectively Equation (1), (2) and (3).
Evaluation on iMaterialist test set
As discussed in Section 2.3 we evaluate the classification, segmentation and inference time using the iMaterialist test set. The logged metrics are presented in Table 9.

Evaluation using combination metrics
The combination metrics Classification score (on both datasets), Segmentation score and Speed score as well as the TTO defined in respectively Equation (15), (17), (18) and (21) are shown in Table 10.

Table 8.: Results Maks R-CNN: Client dataset

Run	EMR	Overall acc	Jaccard
Base1	**35.4%**	81.0%	**74.2%**
Base2	31.8%	78.8%	71.5%
Base3	33.2%	79.4%	73.6%
Base4	4.9%	61.7%	41.1%
Full1	25.5%	87.5%	66.8%
Full2	1.3%	83.9%	50.9%
Full3	13.4%	86.4%	60.9%
Full4	29.2%	**88.7%**	69.9%

Table 9.: Results Maks R-CNN: iMaterialist test set

Run	Av_Speed	EMR	Overall acc	Jaccard	mAP	F1
Base1	1.35 secs	28.9%	77.1%	62.2%	0.63	0.51
Base2	1.39 secs	23.2%	74.6%	59.1%	0.62	0.50
Base3	1.34 secs	23.1%	74.7%	60.8%	0.65	0.52
Base4	**0.66 secs**	14.3%	69.4%	39.8%	0.09	0.09
Full1	1.39 secs	30.3%	88.1%	61.1%	0.62	0.52
Full2	1.36 secs	32.3%	88.1%	57.9%	0.58	0.51
Full3	1.35 secs	36.7%	89.4%	63.5%	0.65	0.54
Full4	1.37 secs	**37.9%**	**89.6%**	**67.7%**	**0.71**	**0.56**

5.5. Final model

The final model is based on the Mask R-CNN architecture. It is chosen based on the results shown in Table 10 and in particular the highest TTO, previously defined as the best model. Experiment Full4 has the highest scores of all experiments performed with the full set of eleven labels. We achieve the overall highest Classification and Segmentation score on the iMaterialist test set as well as Total Trade-Off metric. The Full4 model is trained according to the training scheme and hyperparameters described in Section 4.3.

The TTO of this final model is 1.85, calculated with Equation (21). On the client dataset in 29.2% of the images the classification matches the true labels exactly. 88.7% of the labels are accurately predicted and the JSS is 69.9%. On the iMaterialist test set the EMR is 37.9%, the label accuracy 89.6% and the JSS 67.7%. The mAP is 0.71 and the F1-score is 0.56. Some examples of segmented iMaterialist images in Figure 19. Additional examples are shown in Appendix F. The per class accuracy, true and predicted occurrences as well as the total OA are shown in Appendix E. We discuss these result in more detail in Subsection 6.2.2.

Table 10.: Results Maks R-CNN: combination metrics

Run	Class_C	Class_M	Segmentation	Speed	TTO
Base1	**0.56**	0.48	0.57	0.14	1.75
Base2	0.53	0.44	0.56	0.14	1.75
Base3	0.54	0.44	0.59	0.13	1.71
Base4	0.26	0.32	0.09	**0.07**	0.74
Full1	0.50	0.50	0.57	0.14	1.68
Full2	0.32	0.50	0.55	0.14	1.38
Full3	0.41	0.54	0.60	0.14	1.63
Full4	0.53	**0.56**	**0.64**	0.14	**1.85**

5.5.1. Example segmentation and results

Some parts are omitted due to confidentiality of the client data

One client image next to the resulting segmentation without and with post-processing is discussed. Originally the image is annotated with 'shirt, blouse', 'top, t-shirt, sweatshirt', 'pants' and two times 'shoe'. The prediction, however, is 'top, t-shirt, sweatshirt', 'pants', 'dress' and four times 'shoe'. The two misclassifications are:

- 'shirt, blouse' is classified as 'dress';
- the model classifies the pants area both as pants as well as shoes, having in total four instead of two predicted 'shoe' labels.

The first mistake is one that every person looking at this picture might make as the shirt is very long. To make up for the second mistake, we implemented a simple post-processing scheme where we limit the total occurrence of a class in an image to 2, suppressing the extra objects with lowest classification confidence. If there are more than two shoes in one image with confidence, we ignore the ones that are not the two with the highest confidence score. Table 11 presents the classification metrics of the image. We illustrate that the image looks perfectly classified, but is in fact not an exact match with the ground truth.

Table 11.: Multi-label classification quality of example in Figure **??**

	0	1	2	3	4	5	6	7	8	9	10	11	Metric
True	1	1	0	0	1	0	0	0	1	0	0	0	
Predicted	0	1	0	0	1	0	1	0	1	0	0	0	
True/False	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	
Label accuracy													81.8%
EMR													0%
Jaccard similarity													60%

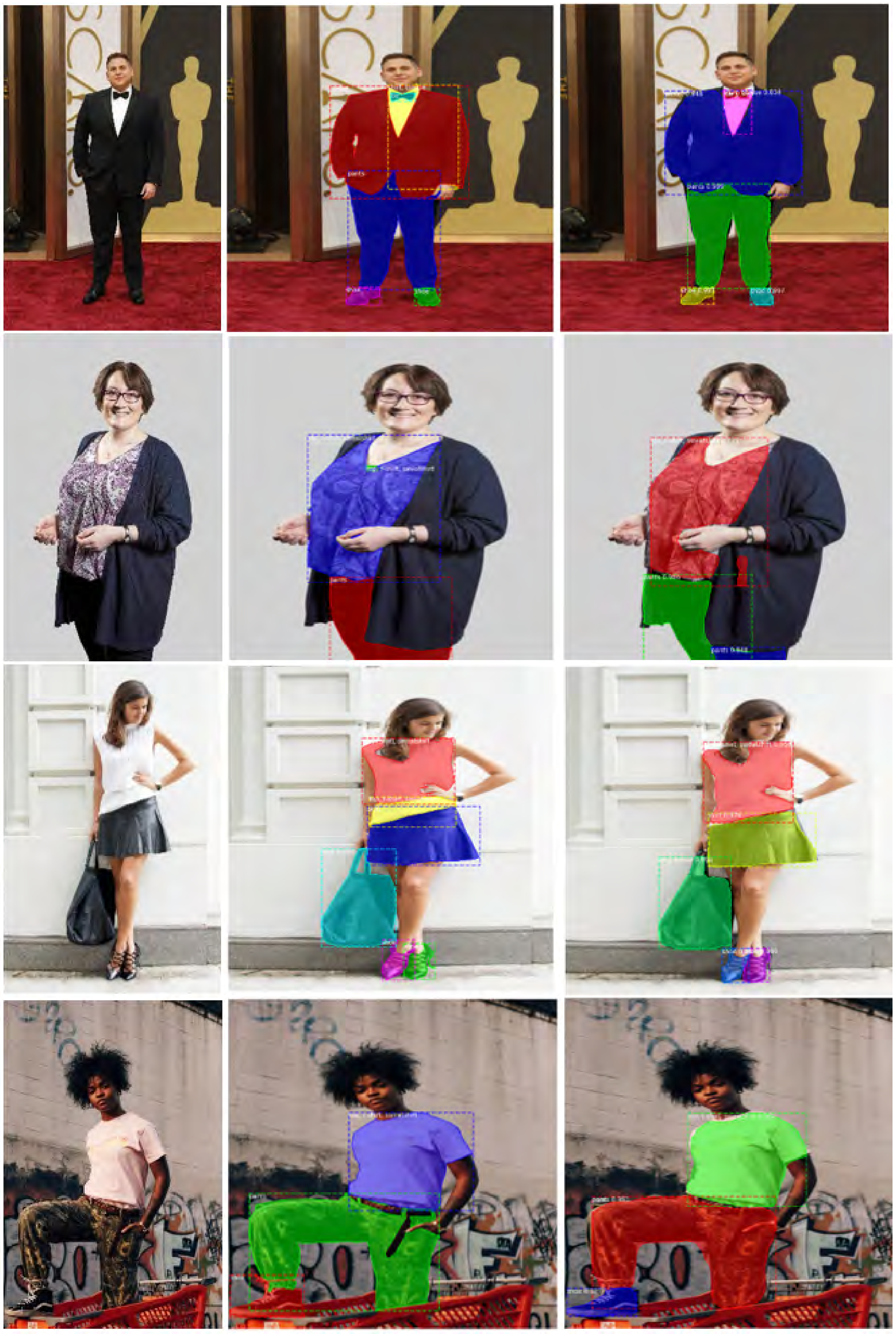


Figure 19.: Final model: examples original-true-segmentation M_{test}

6. Analysis and Discussion

The conducted research is analysed by answering sub-questions presented in Chapter 1. These answers are shown in Section 6.1. We discuss the Experimental Results in Section 6.2.

6.1. Analysis based on sub-questions

Below is the list of answers to the six sub-questions shown in Section 1.3.

Answer to sub-question 1 - Which computer vision approaches are best suited for the task at hand and what are the corresponding, most prominent methods for implementation?
In Chapter 2 we looked in the literature for computer vision approaches and techniques suited for the task at hand. Among the approaches, multi-class image classification, object detection, semantic segmentation and instance segmentation were considered. Those differ in objective, needed train data and result. Segmentation is the most complex approach, but it also gives the most detailed results - per image classification, bounding boxes and segmentation masks of each object. This is why we selected the two image segmentation approaches. The most prominent implementation techniques are RefineNet and Mask R-CNN for semantic and instance segmentation respectively. Both architectures have shown state-of-the-art performance on various datasets. The two architectures are presented in detail, together with some background common mechanisms and deep learning background in Chapter 3.

Answer to sub-question 2 - Which and how much data is needed to train the high-level segmentation models and how should this data be pre-processed?
A segmentation algorithm labels each pixel and thus needs pixel annotated images to train on. There are not many large scale fashion pixel annotated datasets as it is extremely time consuming to collect and annotate each segment in an image. However, we were able to find the iMaterialist dataset that was released in 2019 and contains 45.195 images. We found in the literature that we need at least a thousand images per class to be able to learn the class characteristics.

The number of images needed is dependant on the amount of classes that need to be learned, thus first step of the pre-processing is the selection of classes that are relevant for the client and well represented in the dataset. The base subset used for initial experimenting contains the five most common classes in the dataset. The extended subset contains six additional classes influenced by the class annotations of the client dataset, and overlapping classes from the used pre-trained weights. The following steps include the resizing of each image to a uniform square size of 512 by 512 pixels, the cleaning of the data by relabeling of a wrongly labeled segment, the normalizing of the RGB pixel values and lastly, augmentation. Augmentation is used as both regularization method and to compensate for the real life environment. We use the carefully chosen augmentation techniques Gaussian Blur, Additive Gaussian Noise, Addition and Horizontal flip according to a scheme described in Subsection 4.2.5. The test instances are only resized to improve the inference time.

Answer to sub-question 3 - Which data should be logged while training and testing in order to measure the performance of the models?
In order to compare the performance of the chosen algorithms we decided to report the precision in classification and segmentation, and inference time of each model as explained in Section 2.3. Well known measures for multi-label classification are the exact match ratio, overall accuracy and Jaccard similarity score. We log the true and predicted class labels for each image and experiment for the calculation of the three metrics. The measures we use to quantify the segmentation are mean Average Precision and F1-score for which we use the true and predicted pixel labels. In order to measure the speed, we log the average time it takes each model to classify and segment one image in seconds. We store the weights of each trained model together with the corresponding settings.

Answer to sub-question 4 - Is the complexity of a model architecture related to its performance in terms of the classification and segmentation quality and inference time? If yes, how?
We defined the complexity of a model as its depth and width (see Subsection 2.1.2) and discussed in Section 2.1.2 that the depth of the backbone architechture used, is of significant relevance. The official Mask R-CNN paper shows that the deeper the ResNet backbone, the more accurate the model [He et al., 2017]. However, there is also a trade-off between depth and classification and segmentation quality and given that the architecture with a more complex backbone infers fast enough for our time constraint, it is the preferred one.

Answer to sub-question 5 - Is inference time of under one second for classifying and segmenting the fashion items of one image achievable?
A constraint to the final model is that it needs to infer in less than two seconds. This constraint has to do with it being used in real life at shopping malls for customers there who will take their picture and expect recommendations within a couple of seconds. Therefore, a large advantage of a model to infer in less than two seconds without significant loss in accuracy. This speed

limit and optimization also have to be performed without using additional computational power, comparable to the real life deployment environment.

We found that besides the mentioned above, model complexity, the image size has the largest impact on the inference time, followed by the maximum number of detection instances for proposal-based architectures. An image resized to 128 by 128, 256 by 256, 512 by 512 or 1024 by 1024 pixels infers in on average 0.1, 1.0, 1.7 or 4.0 seconds, respectively. Thus the size 1024 by 1024 makes inference too slow, but all three of the considered smaller image sizes satisfy the speed condition. We found that an image size of 256 by 256 pixel and lower infers within one second, but this there is a significant loss in prediction quality. A decrease of maximum number of detection instances from 100 to 10 reduces the inference time form 2.1 to 1.5 second for an image of 512 by 512 pixels.

Answer to sub-question 6 - Which model performs best on the iMaterialist test set and Client dataset based on the chosen performance measures?
We have selected two model architectures and trained them using pre-processed data described above, various combinations of default and tuned settings and hyperparameters, experimented with different backbone architectures and experimented with utilizing pre-trained weights. As shown in Chapter 5 the implementation of RefineNet that we used failed to produce usable results. However, we find an optimal trade-off between the quality of classification and segmentation and the inference time using the Mask R-CNN instance segmentation architechture. This makes it clear that Mask R-CNN is the preferred architecture.

We found that the model referred to as Full4, has the highest Total Trade-off score (see Table 10), what we predefined as best in Section 5.2. Full4 is trained using the Mask R-CNN architechture. The training details are:

- the above mentioned pre-processing of the data,
- making use of pre-trained COCO weights,
- using a backbone architecture of ResNet with depth 101,
- using a learning rate decay schedule starting with α =0.0005,
- training for 28 epochs,
- using a mini-batch of 2 images per GPU.

This final model is able to classify the extended subset of eleven fashion items being 'shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'jacket', 'pants', 'skirt', 'dress', 'tie', 'shoe', 'bag, wallet' and 'umbrella', and meet the time restriction, when inferred in comparable to the real-life environment, using only CPU computing power.

6.2. Discussion

We discuss the Experimental Results of both RefineNet and Mask R-CNN in Section 6.2.1 and 6.2.2, respectively. The main difference between the two architectures is possibility to pixel-wise segment each object and also extract each object separately without any background using the extra mask head of Mask R-CNN. This is not possible by semantic segmentation.

6.2.1. Discussion of results RefineNet

The no free lunch theorem for machine learning states that no one algorithm is universally best and the real key is to find the optimal task-algorithm pair [Wolpert and Macready, 1997]. This has been proven again by our research. Our initial first choice model was RefineNet, but even though it has performed well on other datasets and problems, it showed ineffective for this project. This could have to do with multiple factors such as implementation, usage of Transfer learning and complexity of the dataset. Firstly, the implementation was found on Git and adapted to work on the iMaterialist dataset. Even though both comments on the repository as well as Bright Cape senior data scientists confirmed that the algorithm should work properly, it resulted in labeling all pixels as background. This result scores high pixel accuracy as most of the pixels contain no fashion items, but is of no use. Secondly, we did not find pre-trained weights for the RefineNet architecture in the needed format. This means that it has to learn the whole natural context as well as fashion items from scratch and gives it a disadvantage compared to Mask R-CNN where we did use Transfer learning. Thirdly, there is the complexity of the dataset. RefineNet has shown state-of-the-art results on for instance, Cityscapes, ADE20K, the object parsing Person-Parts dataset. We believe that the fact that the fashion items are small, detailed and can often be amorphous could be a reason for RefineNet being unable to classify more than background.

6.2.2. Discussion of results Mask R-CNN

First and foremost, the final model using the Mask R-CNN architecture performs above expectations. The initial experiments with the base set were very promising and we felt confident to train on the extended set of eleven classes. The final model shows very good results in terms of multi-label image classification. The Classification scores on both test sets is higher than 0.5, which is excellent considering the harshness of the metric and the somewhat inconsistent annotations. The resulting segments that we inspected look very neat and cover most of the pixels of the objects in an image (see Figure 19 and **??**). Using a carefully selected set of settings for the model and resized images we manage to meet the inference time restriction of (well) under 2 seconds inferring on a standard laptop.

What we see in general is that the model result contains more segments than the actual number of segments annotated. This has two reasons. One, it can be explained by the model not separating foreground from background. We see that it often segments out a jacket on a hanger or a staff member in the background. This, however, is a small issue given the fact that the fashion segmentation model is going to be used in combination with a person detection model. The most important items are the ones that the front person is wearing are classified correctly and that the masks capture the items well. That is indeed the fact in most of the cases. The second reason is that some of the classes are not annotated but predicted. This means that even though the Client dataset does not contain bag labels, our final model is able to segment the bags. This, formally, is a false prediction, but looking at the prediction we can see that in most cases it is spot on.

The largest issue we see is that there are images where the model does not segment an upperbody fashion item at all. This is the case when the contrast with the background is too low or the model does not recognize the item as one of the classes it is trained to know. Luckily it is a very rare occasion. Some of the other shortcomings of the model are mistaking:

- long shirts with dresses;
- oversized shirts worn with nothing under or buttoned up with jackets;
- shirts with tops.

This too, is solvable by post-processing the results. The disadvantage of using post-processing based on hard coded rules is that the model becomes less robust.

7. Conclusion

In this chapter we give answer to the main research question, discuss the limitations within the research and give suggestions for further research.

7.1. Answer to Research Question

In order to conclude this research we answer the main question:

'Which deep learning algorithm performs best on the fashion segmentation and prediction task?'

We find an optimal trade-off between the quality of classification and segmentation, and inference time using the Mask R-CNN instance segmentation architechture. Our final model performs above expectations on both client and iMaterialist test sets. It is able to segment each class of the extended subset of eleven classes being 'shirt, blouse', 'top, t-shirt, sweatshirt', 'sweater', 'jacket', 'pants', 'skirt', 'dress', 'tie', 'shoe', 'bag, wallet' and 'umbrella', and has fast inference time.

This model classifies 29.2% of the images of the client dataset exactly, 88.7% of the labels are accurately predicted and 69.9% ratio of intersection and union of the true and predicted labels. On the iMaterialist dataset, the Exact Match Ratio is 37.9%, the label accuracy 89.6% and the Jaccard similarity score 67.7%. Next, the models shows high mean Average Precision and F1-score, 0.71 and 0.56, respectively. This indicates good purity and completeness of the segmentation.

The final model infers an RGB image of 512 by 512 pixels in 1.37 seconds, using only the CPU of a computer comparable to the one in the real-life environment. The relatively high combination metrics Classification and Segmentation score indicate that the trained model is well capable of performing the task at hand. An inspection of the visualized results confirms this. The masks look very precise and smooth and the assigned classes are correct in most cases.

7.2. Limitation within the research

The main limitations of this research are:

- availability of pixel annotated data;
- resources available for training.

First, there are less than ten public, large scale and pixel annotated datasets that include fashion items. Among those, some consist of small gray-scale images. Even though we are grateful that there are companies investing in developing these datasets and manually annotating thousands of images and millions of segments, the diversity of fashion items, poses, surroundings and people can be improved.

Second, we consider resources a limitation. Both RefineNet and Mask R-CNN algorithm, presented in Chapter 3 involve stochastic optimization and random initialization. If we had unlimited resources we would repeat each run 30 times on each dataset and compare the performance among models based on the average and standard deviation of the 30 repetitions. This seemed unrealistic given that training one run with a sample of only 10% of the iMaterialist train set for 30 epochs on AWS takes 24 hours and costs €34 and we had over 10 unproductive experiments using RefineNet and needed to find the right configurations and tune the hyperparameters of the Mask R-CNN architecture first using the same resources.

7.3. Suggestions for further research

The suggestions for further research are to:

- include training images of population minorities;
- include training images of traditional cultural clothing;
- extract fabric information from the resulting segments.

The iMaterialist dataset is very diverse in many aspects. However, it lacks representation of some population groups. An example would be disabled people, such as people with in a wheelchair, with crutches or prosthetic body parts, which might confuse the model as it is now. Another example is people wearing traditional cultural clothing, for instance a sari, kimono, poncho, pollera, or huipil, which can be any combination of tops, dresses, jackets, skirts and headgear.
The segments that the model we implemented puts out are cut-outs from the images that consist of the classified clothing items. The CP&R project already uses those to extracts the color of this fashion items. An extra possibility is to also classify the fabric it is made of. For instance denim, leather or silk.

Bibliography

[Abdulla, 2017] Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. Retrieved 5 April 2020, from `https://github.com/matterport/Mask_RCNN`.

[Adelson, 2001] Adelson, E. (2001). On seeing stuff: the perception of materials by humans and machines. In *Human Vision and Electronic Imaging VI*, pages 1–12. SPIE.

[Ballard and Brown, 1982] Ballard, D. H. D. H. and Brown, Christopher M., . (1982). *Computer vision*. Englewood Cliffs, N.J.: Prentice-Hall.

[Bright Cape, nd] Bright Cape (n.d.). Homepage | Bright Cape. Retrieved 3 February 2020, from `https://brightcape.nl/`.

[Caesar et al., 2018] Caesar, H., Uijlings, J., and Ferrari, V. (2018). COCO-Stuff: Thing and Stuff Classes in Context. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1209–1218.

[Chen et al., 2016a] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016a). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:834–848.

[Chen et al., 2016b] Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R., and Yuille, A. (2016b). Detect what you can: Detecting and representing objects using holistic models and body parts. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1971–1978.

[Cordts et al., 2016] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. Retrieved 6 April 2020, from `https://arxiv.org/pdf/1604.01685.pdf`.

[Cubuk et al., 2018] Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. (2018). AutoAugment: Learning Augmentation Policies from Data. *ArXiv*, abs/1805.09501.

[de Best, 2019] de Best, R. (2019). Online shopping in the Netherlands - Statistics & Facts. Retrieved 19 February 2020, from `http://www.808multimedia.com/winnt/kernel.htm`.

[DELL, nd] DELL (n.d.). Latitude 5580. Retrieved 3 June 2020, from `https://www.dell.com/support/home/en-us/product-support/servicetag/0-SUxNb2tobzdYSmdQd0ZMbWFNTlUxUT090/overview`.

[Girshick, 2015] Girshick, R. (2015). Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448.

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved 7 April 2020, from `http://www.deeplearningbook.org`.

[Guangyu Zeng, 2018] Guangyu Zeng (2018). An introduction to evaluation metrics for object detection. Retrieved 5 June 2020, from `https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/`.

[Harper, 2019] Harper, C. (2019). Detection, semantic segmentation, instance segmentation. Retrieved 3 March 2020, from `https://slideplayer.com/slide/17627168/#.Xl-GR23YDko.gmail`.

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[Hsu et al., 2018] Hsu, Y.-C., Xu, Z., Kira, Z., and Huang, J. (2018). Learning to Cluster for Proposal-Free Instance Segmentation. Retrieved 6 May 2020, from `https://arxiv.org/pdf/1803.06459.pdf`.

[iMaterialist Fashion Competition group, 2019] iMaterialist Fashion Competition group (2019). iMaterialist (Fashion) 2019 at FGVC6. Retrieved 20 January 2020, from `https://www.kaggle.com/c/imaterialist-fashion-2019-FGVC6/overview`.

[Jung et al., 2020] Jung, A. B., Wada, K., Crall, J., Tanaka, S., Graving, J., Reinders, C., Yadav, S., Banerjee, J., Vecsei, G., Kraft, A., Rui, Z., Borovec, J., Vallentin, C., Zhydenko, S., Pfeiffer, K., Cook, B., Fernández, I., De Rainville, F.-M., Weng, C.-H., Ayala-Acevedo, A., Meudec, R., Laporte, M., et al. (2020). imgaug. Retrieved 5 March 2020, from `https://github.com/aleju/imgaug`.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. Retrieved 6 April 2020, from `https://arxiv.org/abs/1412.6980`.

[Kirillov et al., 2019] Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. (2019). Panoptic Segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9396–9405.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Le, 2018] Le, J. (2018). The 5 Computer Vision Techniques That Will Change How You See The World. Retrieved 10 February 2020, from `https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b`.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lin et al., 2017a] Lin, G., Milan, A., Shen, C., and Reid, I. (2017a). RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5168–5177.

[Lin et al., 2017b] Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017b). Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944.

[Long et al., 2015a] Long, J., Shelhamer, E., and Darrell, T. (2015a). Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

[Long et al., 2015b] Long, J., Shelhamer, E., and Darrell, T. (2015b). Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

[Nair, 2019] Nair, A. (2019). Hands-On Guide To Multi-Label Image Classification With Tensorflow Keras. Retrieved 2 March 2020, from `https://analyticsindiamag.com/multi-label-image-classification-with-tensorflow-keras/`.

[Neven et al., 2019] Neven, D., Brabandere, B. D., Proesmans, M., and Gool, L. V. (2019). Instance Segmentation by Jointly Optimizing Spatial Embeddings and Clustering Bandwidth. Retrieved 11 May 2020, from `https://arxiv.org/pdf/1906.11109.pdf`.

[NotebookCheck, nda] NotebookCheck (n.d.a). NVIDIA GeForce 940MX. Retrieved 22 July 2020, from `https://www.techpowerup.com/gpu-specs/geforce-940mx.c2797`.

[NotebookCheck, ndb] NotebookCheck (n.d.b). NVIDIA Tesla M40. Retrieved 22 July 2020, from `https://www.techpowerup.com/gpu-specs/tesla-m40.c2771`.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc.

[Rosenblatt, 1957] Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton (Project Para). Cornell Aeronautical Laboratory. Retrieved 20 May 2020, from `https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf`.

[Shih, 2010] Shih, F. (2010). *Image Processing and Pattern Recognition: Fundamentals and Techniques*. Wiley.

[Sorower, 2010] Sorower, M. (2010). A Literature Survey on Algorithms for Multi-label Learning. *Oregon State University, Corvallis*, 18:1–25.

[Statista, 2020] Statista (2020). Fashion: Netherlands. Retrieved 12 February 2020, from `https://www.statista.com/outlook/244/144/fashion/netherlands?currency=eur`.

[Thorburn, 1915] Thorburn, W. M. (1915). Occam's Razor. *Mind*, 24(2):287–288.

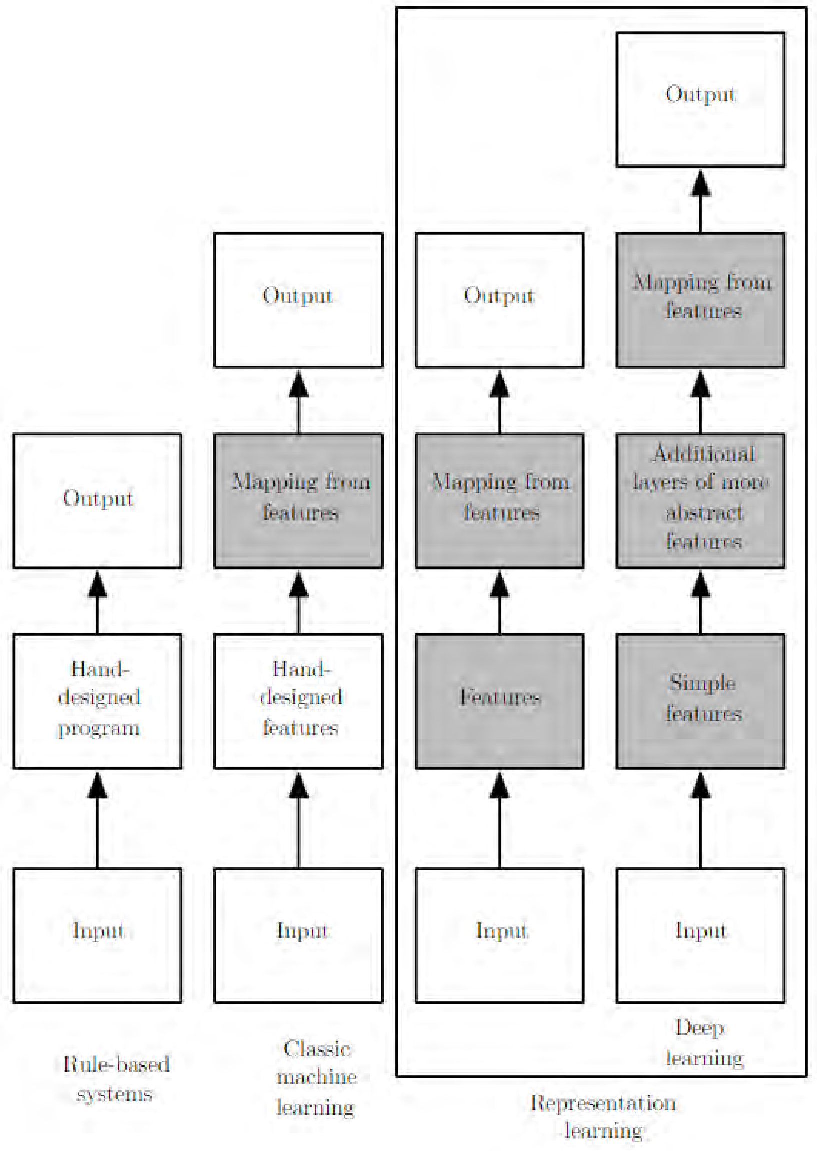
[van Opbroek et al., 2015] van Opbroek, A., Ikram, M. A., Vernooij, M. W., and de Bruijne, M. (2015). Transfer Learning Improves Supervised Image Segmentation Across Imaging Protocols. *IEEE Transactions on Medical Imaging*, 34(5):1018–1030.

[Wang et al., 2016] Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). CNN-RNN: A Unified Framework for Multi-label Image Classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

A. Appendix Approaches to AI

The AI disciplines Rule-based systems, Classic machine learning and Representation learning and the flow through the different parts of each systems. The components that are learned rather than hand-designed are shaded [Goodfellow et al., 2016].



B. Appendix Configurations Mask R-CNN

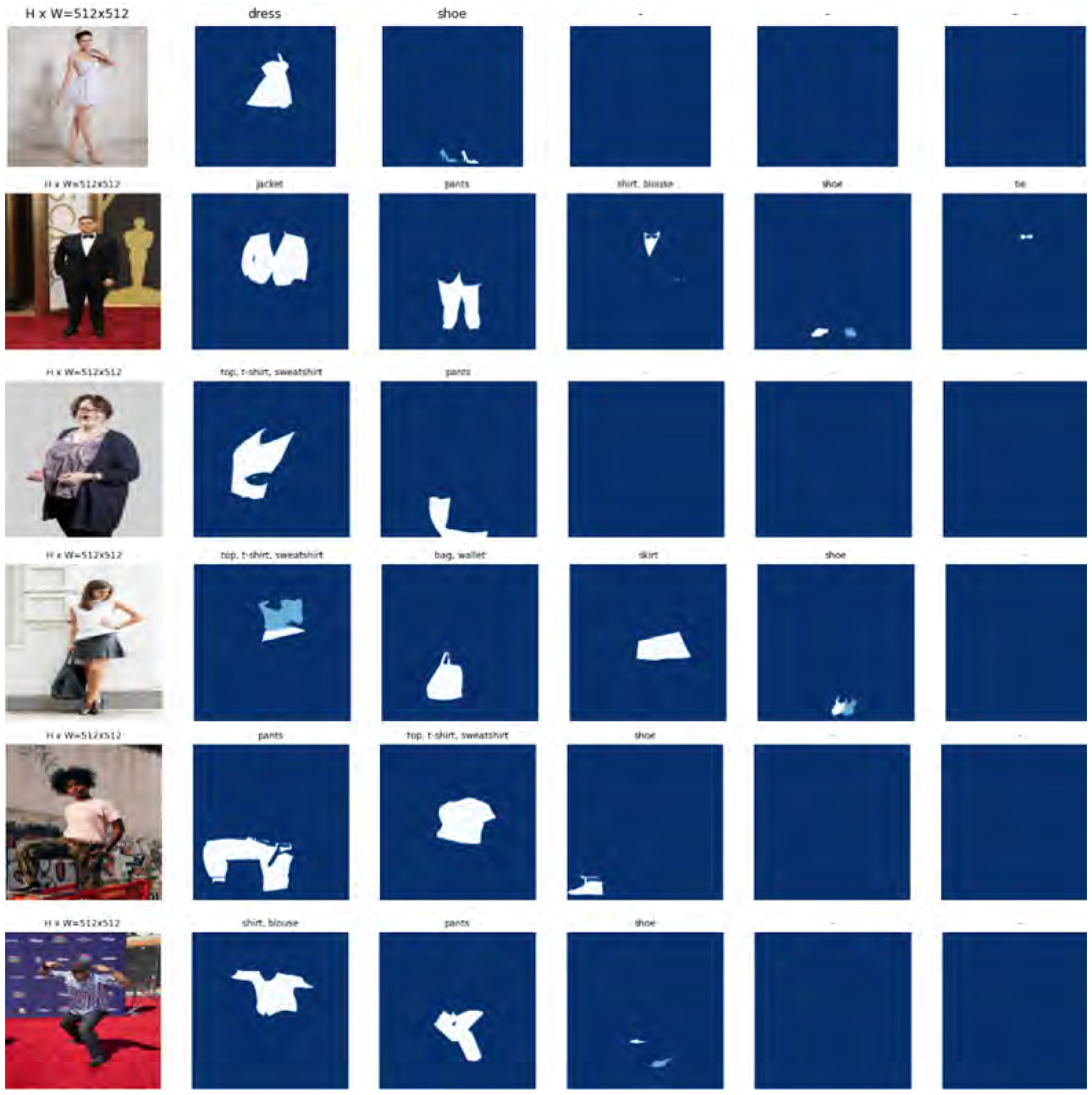
The full list of configurations of the Mask R-CNN algorithm.

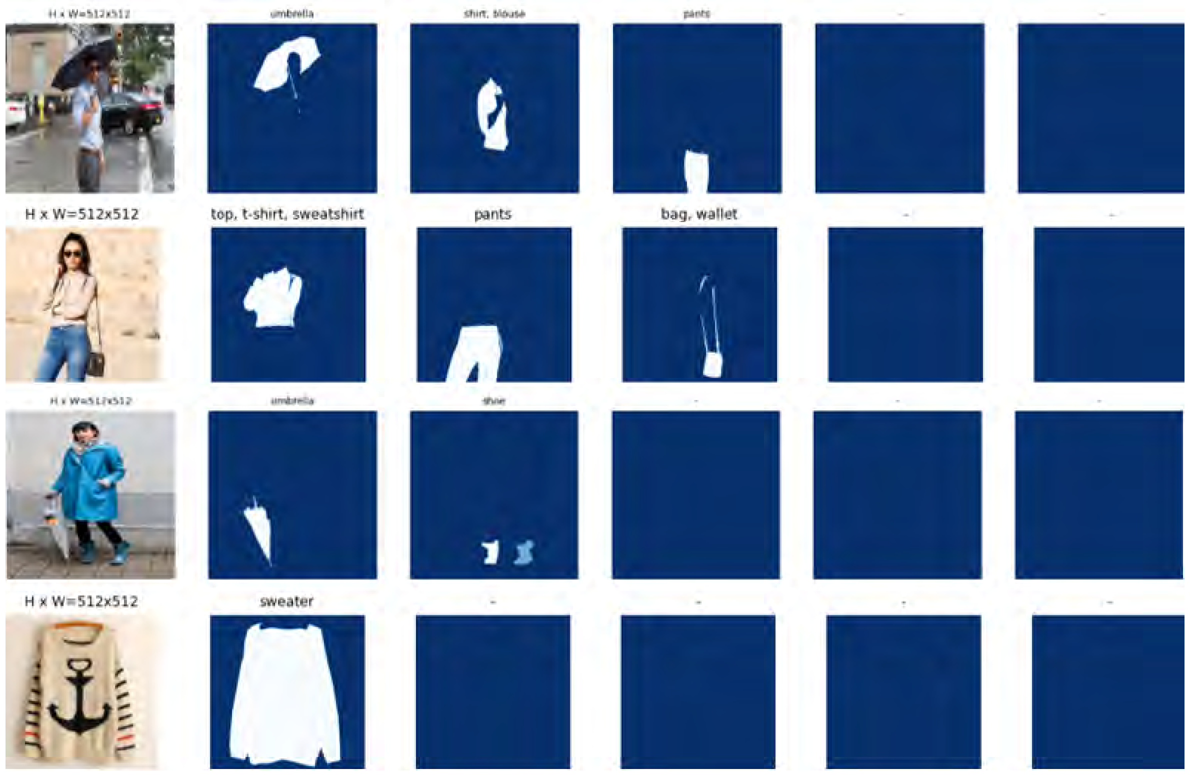
```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     2
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 2
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  512
IMAGE_META_SIZE                18
IMAGE_MIN_DIM                  512
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [512 512   3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0,
                                'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0,
                                'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
```

```
NAME                          fashion
NUM_CLASSES                   6
POOL_SIZE                     7
POST_NMS_ROIS_INFERENCE       1000
POST_NMS_ROIS_TRAINING        2000
PRE_NMS_LIMIT                 6000
ROI_POSITIVE_RATIO            0.33
RPN_ANCHOR_RATIOS             [0.5, 1, 2]
RPN_ANCHOR_SCALES             (16, 32, 64, 128, 256)
RPN_ANCHOR_STRIDE             1
RPN_BBOX_STD_DEV              [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD             0.7
RPN_TRAIN_ANCHORS_PER_IMAGE   256
STEPS_PER_EPOCH               16846.0
TOP_DOWN_PYRAMID_SIZE         256
TRAIN_BN                      False
TRAIN_ROIS_PER_IMAGE          200
USE_MINI_MASK                 True
USE_RPN_ROIS                  True
VALIDATION_STEPS              4212.0
WEIGHT_DECAY                  0.0001
```

C. Appendix Segments example

Example of original images and the ground truth annotated segments from iMaterialist dataset.




H x W=512x512
umbrella
shirt, blouse
pants
H x W=512x512
top, t-shirt, sweatshirt
pants
bag, wallet
H x W=512x512
umbrella
shoe
H x W=512x512
sweater

D. Appendix Loss plots Mask R-CNN

Train and validation loss L, L_{cls} and L_{mask} of Mask R-CNN architecture during training of experiments Base 1 to 4 and Full 1 to 4.

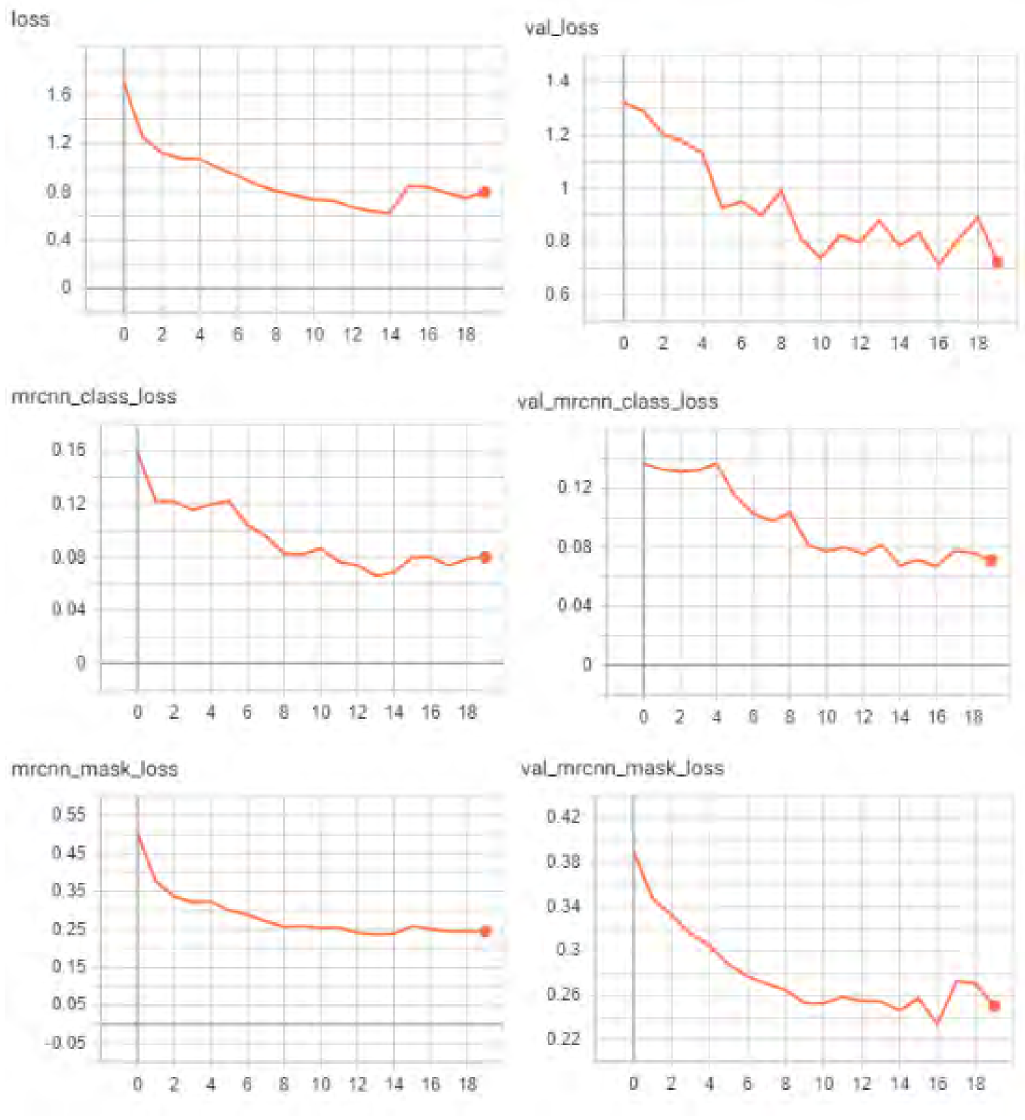


Figure A.1.: Train and validation loss: experiment Base1

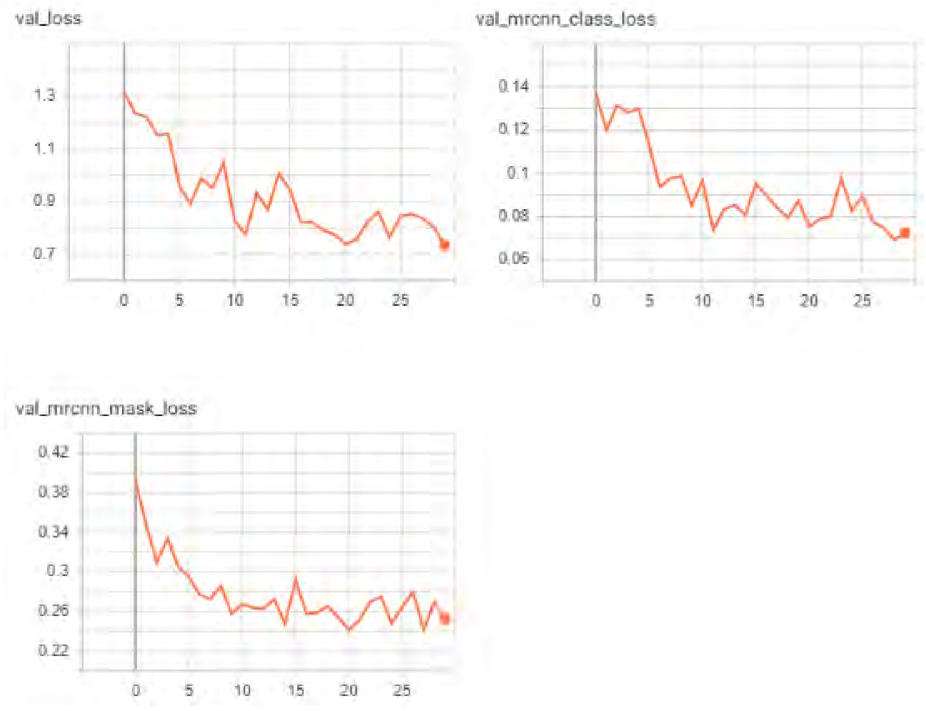


Figure A.2.: Train and validation loss: experiment Base2

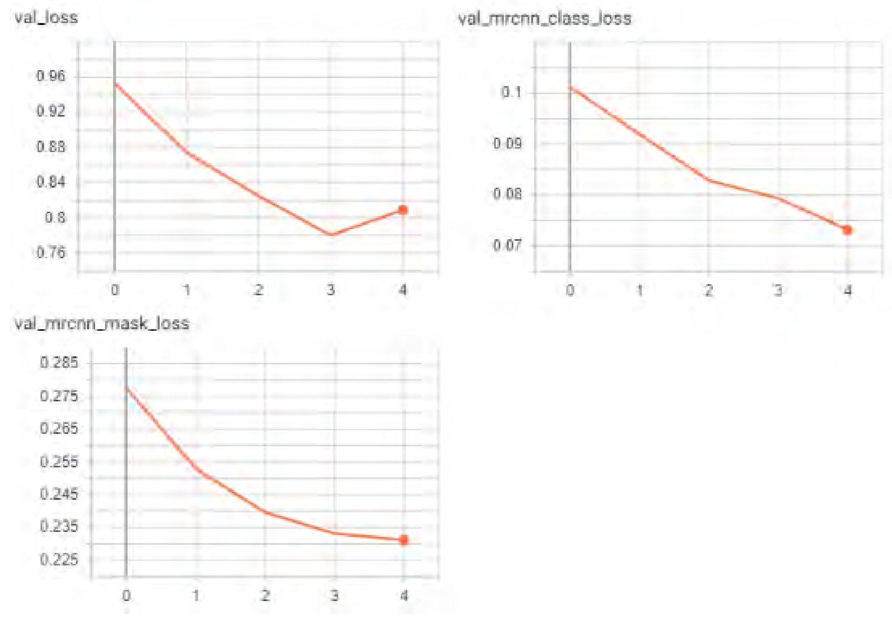


Figure A.3.: Train and validation loss: experiment Base3

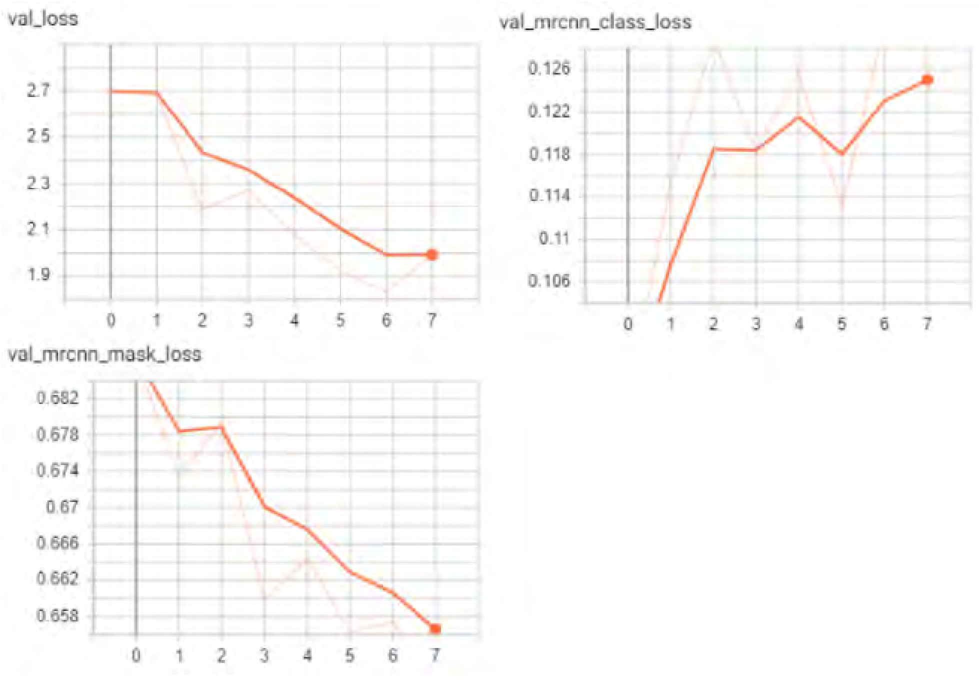


Figure A.4.: Train and validation loss: experiment Base4

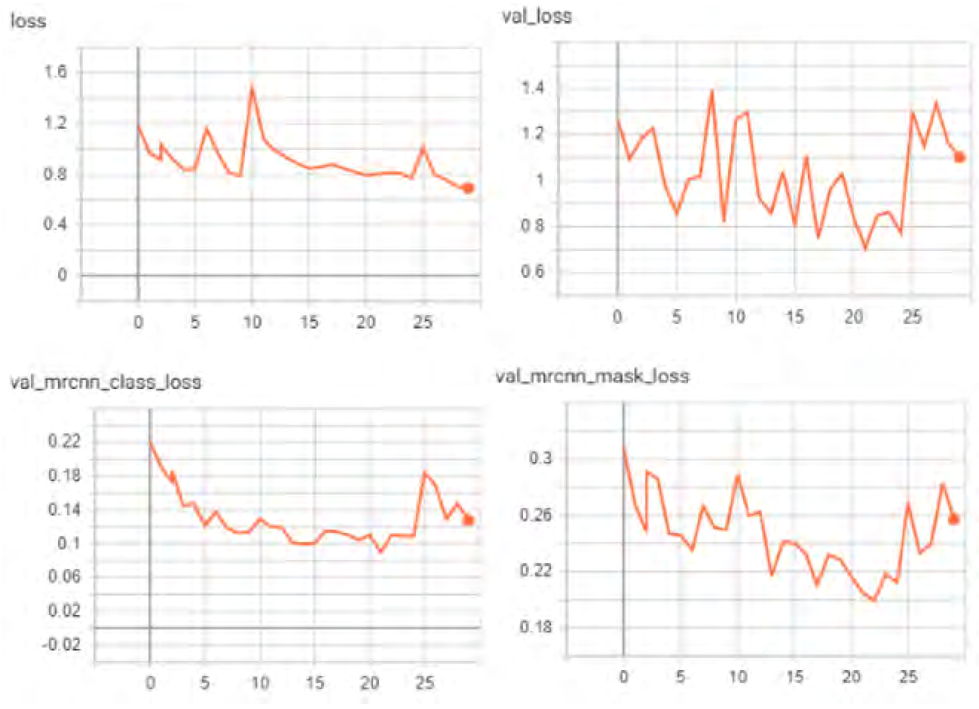


Figure A.5.: Train and validation loss: experiment Full1

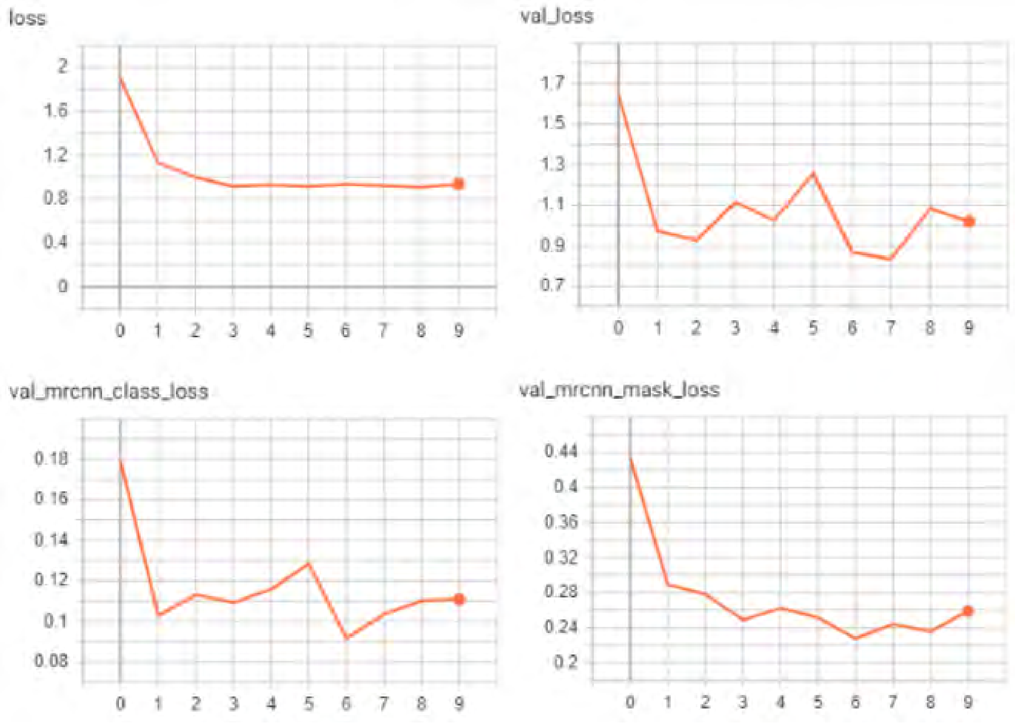


Figure A.6.: Train and validation loss: experiment Full2

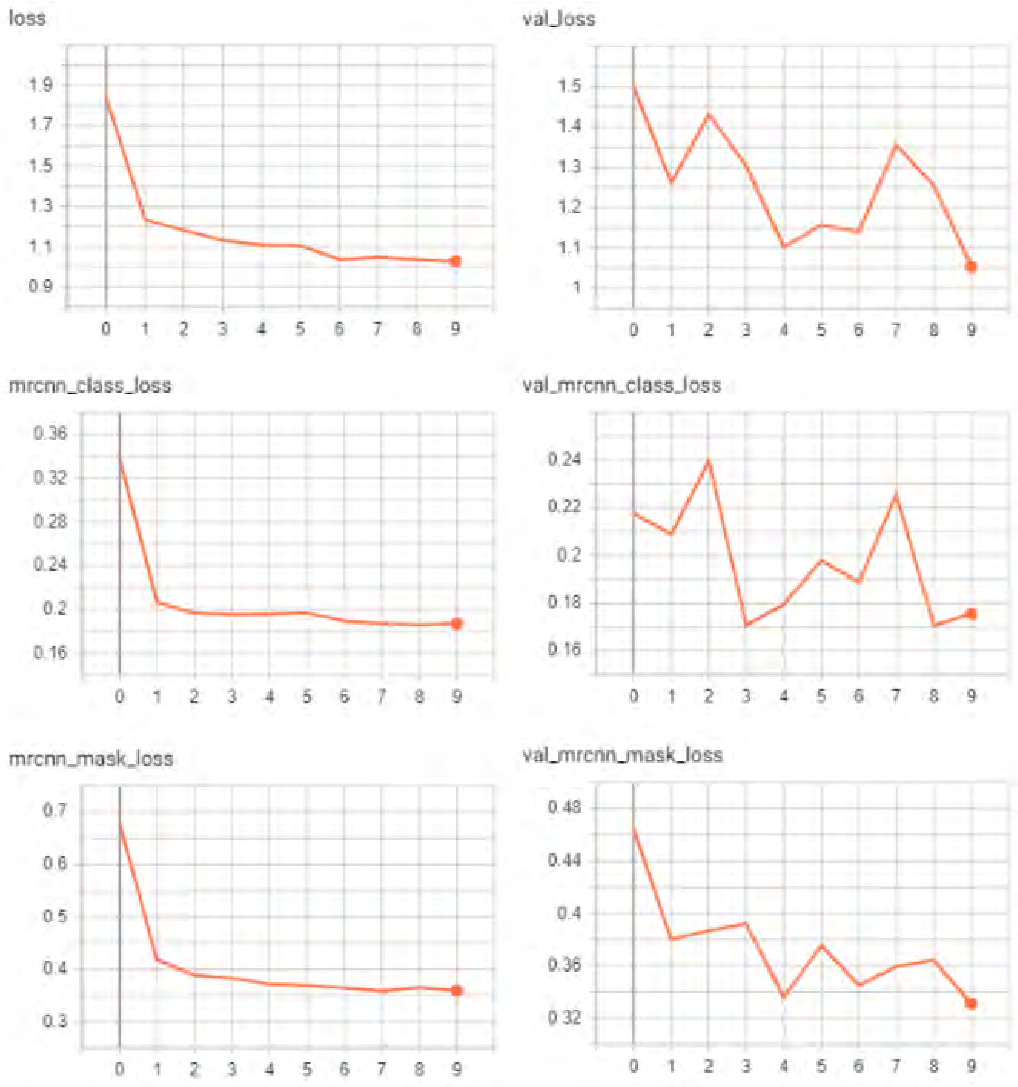


Figure A.7.: Train and validation loss: experiment Full3

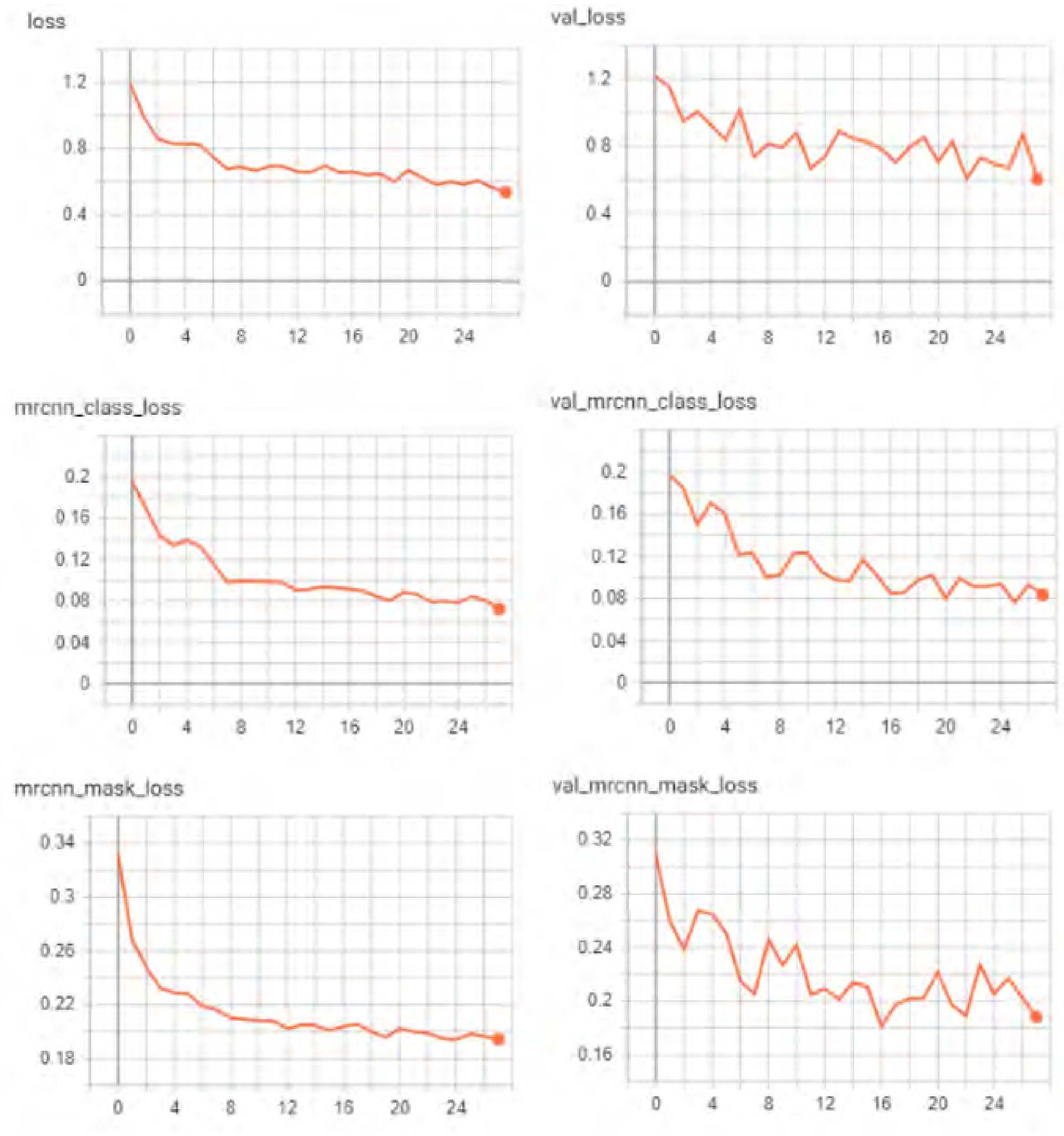


Figure A.8.: Train and validation loss: experiment Full4

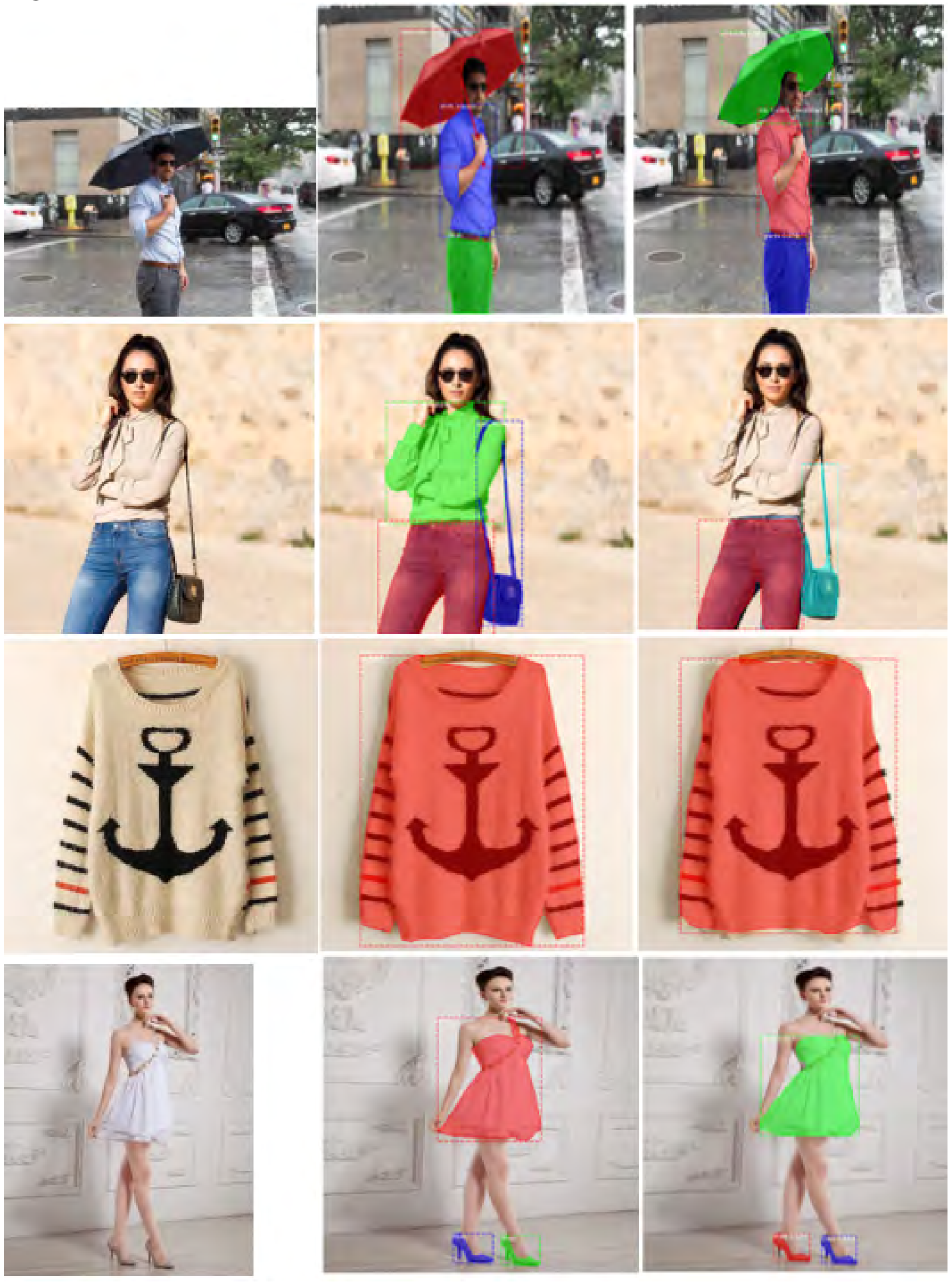
E. Appendix Per class and overall accuracy final model

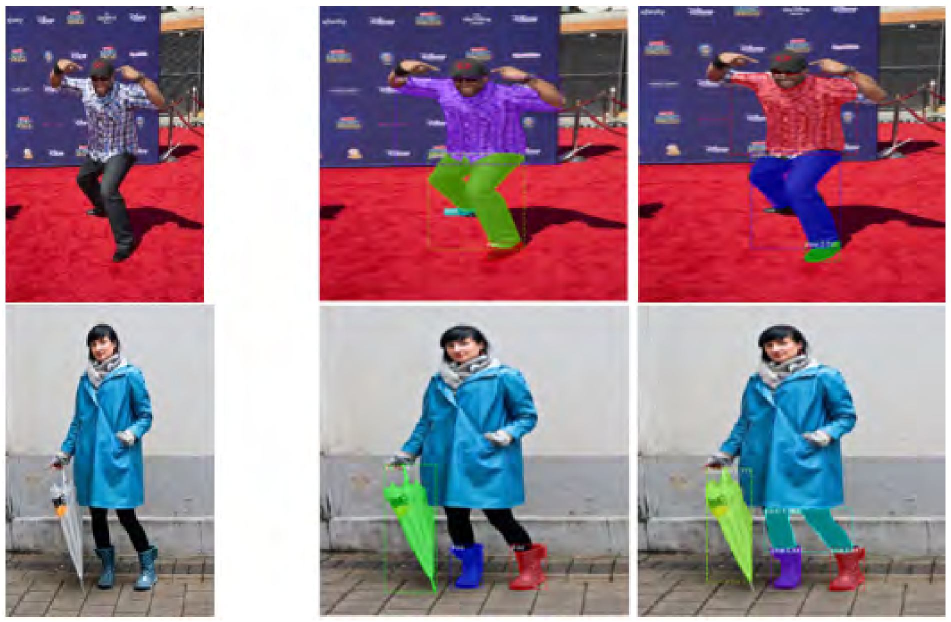
Per class and overall accuracy and label occurrences inferred on the final model needed to determine the quality of multi-label classification.

	Client dataset				iMaterialist	dataset
Class name	Accuracy	True	Predicted	Accuracy	True	Predicted
'shirt, blouse'	80.4%	93	9	88.6%	152	87
'top, t-shirt, sweatshirt'	74.1%	199	249	72.8%	352	451
'sweater'	79.8%	92	0	96.4%	37	2
'jacket'	76.9%	275	260	87.3%	181	225
'pants'	90.5%	291	324	86.3%	273	393
'skirt'	91.9%	93	76	88.3%	117	166
'dress'	92.1%	61	73	86.3%	422	507
'tie'	98.5%	0	7	97.5%	41	55
'shoe'	97.4%	864	919	93.0%	1.038	1.303
'bag, wallet'	93.6%	0	31	88.9%	161	273
'umbrella'	100.0%	0	0	99.8%	3	3
Total	88.7%	1.968	1.948	89.6%	2.777	3.465

F. Appendix Additional results final model

Additional examples of original images, the ground truth annotated segments and resulting segments from iMaterialist dataset.





G. Appendix Model Summary RefineNet

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
data (InputLayer)                (None, 384, 384, 3)   0
____________________________________________________________________________________________________
conv1_zeropadding (ZeroPadding2  (None, 390, 390, 3)   0           data[0][0]
____________________________________________________________________________________________________
conv1 (Conv2D)                   (None, 192, 192, 64)  9408        conv1_zeropadding[0][0]
____________________________________________________________________________________________________
bn_conv1 (BatchNormalization)    (None, 192, 192, 64)  256         conv1[0][0]
____________________________________________________________________________________________________
scale_conv1 (Scale)              (None, 192, 192, 64)  128         bn_conv1[0][0]
____________________________________________________________________________________________________
conv1_relu (Activation)          (None, 192, 192, 64)  0           scale_conv1[0][0]
____________________________________________________________________________________________________
pool1 (MaxPooling2D)             (None, 96, 96, 64)    0           conv1_relu[0][0]
____________________________________________________________________________________________________
res2a_branch2a (Conv2D)          (None, 96, 96, 64)    4096        pool1[0][0]
____________________________________________________________________________________________________
bn2a_branch2a (BatchNormalizati  (None, 96, 96, 64)    256         res2a_branch2a[0][0]
____________________________________________________________________________________________________
scale2a_branch2a (Scale)         (None, 96, 96, 64)    128         bn2a_branch2a[0][0]
____________________________________________________________________________________________________
res2a_branch2a_relu (Activation  (None, 96, 96, 64)    0           scale2a_branch2a[0][0]
____________________________________________________________________________________________________
res2a_branch2b_zeropadding (Zer  (None, 98, 98, 64)    0           res2a_branch2a_relu[0][0]
____________________________________________________________________________________________________
res2a_branch2b (Conv2D)          (None, 96, 96, 64)    36864       res2a_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn2a_branch2b (BatchNormalizati  (None, 96, 96, 64)    256         res2a_branch2b[0][0]
____________________________________________________________________________________________________
scale2a_branch2b (Scale)         (None, 96, 96, 64)    128         bn2a_branch2b[0][0]
____________________________________________________________________________________________________
res2a_branch2b_relu (Activation  (None, 96, 96, 64)    0           scale2a_branch2b[0][0]
____________________________________________________________________________________________________
res2a_branch2c (Conv2D)          (None, 96, 96, 256)   16384       res2a_branch2b_relu[0][0]
____________________________________________________________________________________________________
res2a_branch1 (Conv2D)           (None, 96, 96, 256)   16384       pool1[0][0]
____________________________________________________________________________________________________
```

```
bn2a_branch2c (BatchNormalizati (None, 96, 96, 256)  1024        res2a_branch2c[0][0]
__________________________________________________________________________________________________
bn2a_branch1 (BatchNormalizatio (None, 96, 96, 256)  1024        res2a_branch1[0][0]
__________________________________________________________________________________________________
scale2a_branch2c (Scale)        (None, 96, 96, 256)  512         bn2a_branch2c[0][0]
__________________________________________________________________________________________________
scale2a_branch1 (Scale)         (None, 96, 96, 256)  512         bn2a_branch1[0][0]
__________________________________________________________________________________________________
res2a (Add)                     (None, 96, 96, 256)  0           scale2a_branch2c[0][0]
                                                                 scale2a_branch1[0][0]
__________________________________________________________________________________________________
res2a_relu (Activation)         (None, 96, 96, 256)  0           res2a[0][0]
__________________________________________________________________________________________________
res2b_branch2a (Conv2D)         (None, 96, 96, 64)   16384       res2a_relu[0][0]
__________________________________________________________________________________________________
bn2b_branch2a (BatchNormalizati (None, 96, 96, 64)   256         res2b_branch2a[0][0]
__________________________________________________________________________________________________
scale2b_branch2a (Scale)        (None, 96, 96, 64)   128         bn2b_branch2a[0][0]
__________________________________________________________________________________________________
res2b_branch2a_relu (Activation (None, 96, 96, 64)   0           scale2b_branch2a[0][0]
__________________________________________________________________________________________________
res2b_branch2b_zeropadding (Zer (None, 98, 98, 64)   0           res2b_branch2a_relu[0][0]
__________________________________________________________________________________________________
res2b_branch2b (Conv2D)         (None, 96, 96, 64)   36864       res2b_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn2b_branch2b (BatchNormalizati (None, 96, 96, 64)   256         res2b_branch2b[0][0]
__________________________________________________________________________________________________
scale2b_branch2b (Scale)        (None, 96, 96, 64)   128         bn2b_branch2b[0][0]
__________________________________________________________________________________________________
res2b_branch2b_relu (Activation (None, 96, 96, 64)   0           scale2b_branch2b[0][0]
__________________________________________________________________________________________________
res2b_branch2c (Conv2D)         (None, 96, 96, 256)  16384       res2b_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn2b_branch2c (BatchNormalizati (None, 96, 96, 256)  1024        res2b_branch2c[0][0]
__________________________________________________________________________________________________
scale2b_branch2c (Scale)        (None, 96, 96, 256)  512         bn2b_branch2c[0][0]
__________________________________________________________________________________________________
res2b (Add)                     (None, 96, 96, 256)  0           scale2b_branch2c[0][0]
                                                                 res2a_relu[0][0]
__________________________________________________________________________________________________
res2b_relu (Activation)         (None, 96, 96, 256)  0           res2b[0][0]
__________________________________________________________________________________________________
res2c_branch2a (Conv2D)         (None, 96, 96, 64)   16384       res2b_relu[0][0]
__________________________________________________________________________________________________
bn2c_branch2a (BatchNormalizati (None, 96, 96, 64)   256         res2c_branch2a[0][0]
__________________________________________________________________________________________________
scale2c_branch2a (Scale)        (None, 96, 96, 64)   128         bn2c_branch2a[0][0]
```

```
____________________________________________________________________________________________________
res2c_branch2a_relu (Activation (None, 96, 96, 64)   0           scale2c_branch2a[0][0]
____________________________________________________________________________________________________
res2c_branch2b_zeropadding (Zer (None, 98, 98, 64)   0           res2c_branch2a_relu[0][0]
____________________________________________________________________________________________________
res2c_branch2b (Conv2D)         (None, 96, 96, 64)   36864       res2c_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn2c_branch2b (BatchNormalizati (None, 96, 96, 64)   256         res2c_branch2b[0][0]
____________________________________________________________________________________________________
scale2c_branch2b (Scale)        (None, 96, 96, 64)   128         bn2c_branch2b[0][0]
____________________________________________________________________________________________________
res2c_branch2b_relu (Activation (None, 96, 96, 64)   0           scale2c_branch2b[0][0]
____________________________________________________________________________________________________
res2c_branch2c (Conv2D)         (None, 96, 96, 256)  16384       res2c_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn2c_branch2c (BatchNormalizati (None, 96, 96, 256)  1024        res2c_branch2c[0][0]
____________________________________________________________________________________________________
scale2c_branch2c (Scale)        (None, 96, 96, 256)  512         bn2c_branch2c[0][0]
____________________________________________________________________________________________________
res2c (Add)                     (None, 96, 96, 256)  0           scale2c_branch2c[0][0]
                                                                 res2b_relu[0][0]
____________________________________________________________________________________________________
res2c_relu (Activation)         (None, 96, 96, 256)  0           res2c[0][0]
____________________________________________________________________________________________________
res3a_branch2a (Conv2D)         (None, 48, 48, 128)  32768       res2c_relu[0][0]
____________________________________________________________________________________________________
bn3a_branch2a (BatchNormalizati (None, 48, 48, 128)  512         res3a_branch2a[0][0]
____________________________________________________________________________________________________
scale3a_branch2a (Scale)        (None, 48, 48, 128)  256         bn3a_branch2a[0][0]
____________________________________________________________________________________________________
res3a_branch2a_relu (Activation (None, 48, 48, 128)  0           scale3a_branch2a[0][0]
____________________________________________________________________________________________________
res3a_branch2b_zeropadding (Zer (None, 50, 50, 128)  0           res3a_branch2a_relu[0][0]
____________________________________________________________________________________________________
res3a_branch2b (Conv2D)         (None, 48, 48, 128)  147456      res3a_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn3a_branch2b (BatchNormalizati (None, 48, 48, 128)  512         res3a_branch2b[0][0]
____________________________________________________________________________________________________
scale3a_branch2b (Scale)        (None, 48, 48, 128)  256         bn3a_branch2b[0][0]
____________________________________________________________________________________________________
res3a_branch2b_relu (Activation (None, 48, 48, 128)  0           scale3a_branch2b[0][0]
____________________________________________________________________________________________________
res3a_branch2c (Conv2D)         (None, 48, 48, 512)  65536       res3a_branch2b_relu[0][0]
____________________________________________________________________________________________________
res3a_branch1 (Conv2D)          (None, 48, 48, 512)  131072      res2c_relu[0][0]
____________________________________________________________________________________________________
bn3a_branch2c (BatchNormalizati (None, 48, 48, 512)  2048        res3a_branch2c[0][0]
__________________________________________________________________________________________________
bn3a_branch1 (BatchNormalizatio (None, 48, 48, 512)  2048        res3a_branch1[0][0]
__________________________________________________________________________________________________
scale3a_branch2c (Scale)        (None, 48, 48, 512)  1024        bn3a_branch2c[0][0]
__________________________________________________________________________________________________
scale3a_branch1 (Scale)         (None, 48, 48, 512)  1024        bn3a_branch1[0][0]
__________________________________________________________________________________________________
res3a (Add)                     (None, 48, 48, 512)  0           scale3a_branch2c[0][0]
                                                                 scale3a_branch1[0][0]
__________________________________________________________________________________________________
res3a_relu (Activation)         (None, 48, 48, 512)  0           res3a[0][0]
__________________________________________________________________________________________________
res3b1_branch2a (Conv2D)        (None, 48, 48, 128)  65536       res3a_relu[0][0]
__________________________________________________________________________________________________
bn3b1_branch2a (BatchNormalizat (None, 48, 48, 128)  512         res3b1_branch2a[0][0]
__________________________________________________________________________________________________
scale3b1_branch2a (Scale)       (None, 48, 48, 128)  256         bn3b1_branch2a[0][0]
__________________________________________________________________________________________________
res3b1_branch2a_relu (Activatio (None, 48, 48, 128)  0           scale3b1_branch2a[0][0]
__________________________________________________________________________________________________
res3b1_branch2b_zeropadding (Ze (None, 50, 50, 128)  0           res3b1_branch2a_relu[0][0]
__________________________________________________________________________________________________
res3b1_branch2b (Conv2D)        (None, 48, 48, 128)  147456      res3b1_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn3b1_branch2b (BatchNormalizat (None, 48, 48, 128)  512         res3b1_branch2b[0][0]
__________________________________________________________________________________________________
scale3b1_branch2b (Scale)       (None, 48, 48, 128)  256         bn3b1_branch2b[0][0]
__________________________________________________________________________________________________
res3b1_branch2b_relu (Activatio (None, 48, 48, 128)  0           scale3b1_branch2b[0][0]
__________________________________________________________________________________________________
res3b1_branch2c (Conv2D)        (None, 48, 48, 512)  65536       res3b1_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn3b1_branch2c (BatchNormalizat (None, 48, 48, 512)  2048        res3b1_branch2c[0][0]
__________________________________________________________________________________________________
scale3b1_branch2c (Scale)       (None, 48, 48, 512)  1024        bn3b1_branch2c[0][0]
__________________________________________________________________________________________________
res3b1 (Add)                    (None, 48, 48, 512)  0           scale3b1_branch2c[0][0]
                                                                 res3a_relu[0][0]
__________________________________________________________________________________________________
res3b1_relu (Activation)        (None, 48, 48, 512)  0           res3b1[0][0]
__________________________________________________________________________________________________
res3b2_branch2a (Conv2D)        (None, 48, 48, 128)  65536       res3b1_relu[0][0]
__________________________________________________________________________________________________
bn3b2_branch2a (BatchNormalizat (None, 48, 48, 128)  512         res3b2_branch2a[0][0]
__________________________________________________________________________________________________
scale3b2_branch2a (Scale)       (None, 48, 48, 128)  256         bn3b2_branch2a[0][0]
__________________________________________________________________________________________________
```

```
res3b2_branch2a_relu (Activatio (None, 48, 48, 128)  0           scale3b2_branch2a[0][0]
__________________________________________________________________________________________________
res3b2_branch2b_zeropadding (Ze (None, 50, 50, 128)  0           res3b2_branch2a_relu[0][0]
__________________________________________________________________________________________________
res3b2_branch2b (Conv2D)        (None, 48, 48, 128)  147456      res3b2_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn3b2_branch2b (BatchNormalizat (None, 48, 48, 128)  512         res3b2_branch2b[0][0]
__________________________________________________________________________________________________
scale3b2_branch2b (Scale)       (None, 48, 48, 128)  256         bn3b2_branch2b[0][0]
__________________________________________________________________________________________________
res3b2_branch2b_relu (Activatio (None, 48, 48, 128)  0           scale3b2_branch2b[0][0]
__________________________________________________________________________________________________
res3b2_branch2c (Conv2D)        (None, 48, 48, 512)  65536       res3b2_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn3b2_branch2c (BatchNormalizat (None, 48, 48, 512)  2048        res3b2_branch2c[0][0]
__________________________________________________________________________________________________
scale3b2_branch2c (Scale)       (None, 48, 48, 512)  1024        bn3b2_branch2c[0][0]
__________________________________________________________________________________________________
res3b2 (Add)                    (None, 48, 48, 512)  0           scale3b2_branch2c[0][0]
                                                                 res3b1_relu[0][0]
__________________________________________________________________________________________________
res3b2_relu (Activation)        (None, 48, 48, 512)  0           res3b2[0][0]
__________________________________________________________________________________________________
res3b3_branch2a (Conv2D)        (None, 48, 48, 128)  65536       res3b2_relu[0][0]
__________________________________________________________________________________________________
bn3b3_branch2a (BatchNormalizat (None, 48, 48, 128)  512         res3b3_branch2a[0][0]
__________________________________________________________________________________________________
scale3b3_branch2a (Scale)       (None, 48, 48, 128)  256         bn3b3_branch2a[0][0]
__________________________________________________________________________________________________
res3b3_branch2a_relu (Activatio (None, 48, 48, 128)  0           scale3b3_branch2a[0][0]
__________________________________________________________________________________________________
res3b3_branch2b_zeropadding (Ze (None, 50, 50, 128)  0           res3b3_branch2a_relu[0][0]
__________________________________________________________________________________________________
res3b3_branch2b (Conv2D)        (None, 48, 48, 128)  147456      res3b3_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn3b3_branch2b (BatchNormalizat (None, 48, 48, 128)  512         res3b3_branch2b[0][0]
__________________________________________________________________________________________________
scale3b3_branch2b (Scale)       (None, 48, 48, 128)  256         bn3b3_branch2b[0][0]
__________________________________________________________________________________________________
res3b3_branch2b_relu (Activatio (None, 48, 48, 128)  0           scale3b3_branch2b[0][0]
__________________________________________________________________________________________________
res3b3_branch2c (Conv2D)        (None, 48, 48, 512)  65536       res3b3_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn3b3_branch2c (BatchNormalizat (None, 48, 48, 512)  2048        res3b3_branch2c[0][0]
__________________________________________________________________________________________________
scale3b3_branch2c (Scale)       (None, 48, 48, 512)  1024        bn3b3_branch2c[0][0]
__________________________________________________________________________________________________
```

```
res3b3 (Add)                    (None, 48, 48, 512)  0           scale3b3_branch2c[0][0]
                                                                 res3b2_relu[0][0]
__________________________________________________________________________________________________
res3b3_relu (Activation)        (None, 48, 48, 512)  0           res3b3[0][0]
__________________________________________________________________________________________________
res4a_branch2a (Conv2D)         (None, 24, 24, 256)  131072      res3b3_relu[0][0]
__________________________________________________________________________________________________
bn4a_branch2a (BatchNormalizati (None, 24, 24, 256)  1024        res4a_branch2a[0][0]
__________________________________________________________________________________________________
scale4a_branch2a (Scale)        (None, 24, 24, 256)  512         bn4a_branch2a[0][0]
__________________________________________________________________________________________________
res4a_branch2a_relu (Activation (None, 24, 24, 256)  0           scale4a_branch2a[0][0]
__________________________________________________________________________________________________
res4a_branch2b_zeropadding (Zer (None, 26, 26, 256)  0           res4a_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4a_branch2b (Conv2D)         (None, 24, 24, 256)  589824      res4a_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn4a_branch2b (BatchNormalizati (None, 24, 24, 256)  1024        res4a_branch2b[0][0]
__________________________________________________________________________________________________
scale4a_branch2b (Scale)        (None, 24, 24, 256)  512         bn4a_branch2b[0][0]
__________________________________________________________________________________________________
res4a_branch2b_relu (Activation (None, 24, 24, 256)  0           scale4a_branch2b[0][0]
__________________________________________________________________________________________________
res4a_branch2c (Conv2D)         (None, 24, 24, 1024) 262144      res4a_branch2b_relu[0][0]
__________________________________________________________________________________________________
res4a_branch1 (Conv2D)          (None, 24, 24, 1024) 524288      res3b3_relu[0][0]
__________________________________________________________________________________________________
bn4a_branch2c (BatchNormalizati (None, 24, 24, 1024) 4096        res4a_branch2c[0][0]
__________________________________________________________________________________________________
bn4a_branch1 (BatchNormalizatio (None, 24, 24, 1024) 4096        res4a_branch1[0][0]
__________________________________________________________________________________________________
scale4a_branch2c (Scale)        (None, 24, 24, 1024) 2048        bn4a_branch2c[0][0]
__________________________________________________________________________________________________
scale4a_branch1 (Scale)         (None, 24, 24, 1024) 2048        bn4a_branch1[0][0]
__________________________________________________________________________________________________
res4a (Add)                     (None, 24, 24, 1024) 0           scale4a_branch2c[0][0]
                                                                 scale4a_branch1[0][0]
__________________________________________________________________________________________________
res4a_relu (Activation)         (None, 24, 24, 1024) 0           res4a[0][0]
__________________________________________________________________________________________________
res4b1_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4a_relu[0][0]
__________________________________________________________________________________________________
bn4b1_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b1_branch2a[0][0]
__________________________________________________________________________________________________
scale4b1_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b1_branch2a[0][0]
__________________________________________________________________________________________________
res4b1_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b1_branch2a[0][0]
__________________________________________________________________________________________________
res4b1_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b1_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b1_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b1_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn4b1_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b1_branch2b[0][0]
__________________________________________________________________________________________________
scale4b1_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b1_branch2b[0][0]
__________________________________________________________________________________________________
res4b1_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b1_branch2b[0][0]
__________________________________________________________________________________________________
res4b1_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b1_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn4b1_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b1_branch2c[0][0]
__________________________________________________________________________________________________
scale4b1_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b1_branch2c[0][0]
__________________________________________________________________________________________________
res4b1 (Add)                    (None, 24, 24, 1024) 0           scale4b1_branch2c[0][0]
                                                                 res4a_relu[0][0]
__________________________________________________________________________________________________
res4b1_relu (Activation)        (None, 24, 24, 1024) 0           res4b1[0][0]
__________________________________________________________________________________________________
res4b2_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b1_relu[0][0]
__________________________________________________________________________________________________
bn4b2_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b2_branch2a[0][0]
__________________________________________________________________________________________________
scale4b2_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b2_branch2a[0][0]
__________________________________________________________________________________________________
res4b2_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b2_branch2a[0][0]
__________________________________________________________________________________________________
res4b2_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b2_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b2_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b2_branch2b_zeropadding[0][0]
__________________________________________________________________________________________________
bn4b2_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b2_branch2b[0][0]
__________________________________________________________________________________________________
scale4b2_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b2_branch2b[0][0]
__________________________________________________________________________________________________
res4b2_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b2_branch2b[0][0]
__________________________________________________________________________________________________
res4b2_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b2_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn4b2_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b2_branch2c[0][0]
__________________________________________________________________________________________________
scale4b2_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b2_branch2c[0][0]
__________________________________________________________________________________________________
res4b2 (Add)                    (None, 24, 24, 1024) 0           scale4b2_branch2c[0][0]
```

```
                                                                 res4b1_relu[0][0]
____________________________________________________________________________________________________
res4b2_relu (Activation)        (None, 24, 24, 1024) 0           res4b2[0][0]
____________________________________________________________________________________________________
res4b3_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b2_relu[0][0]
____________________________________________________________________________________________________
bn4b3_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b3_branch2a[0][0]
____________________________________________________________________________________________________
scale4b3_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b3_branch2a[0][0]
____________________________________________________________________________________________________
res4b3_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b3_branch2a[0][0]
____________________________________________________________________________________________________
res4b3_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b3_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b3_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b3_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b3_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b3_branch2b[0][0]
____________________________________________________________________________________________________
scale4b3_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b3_branch2b[0][0]
____________________________________________________________________________________________________
res4b3_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b3_branch2b[0][0]
____________________________________________________________________________________________________
res4b3_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b3_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b3_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b3_branch2c[0][0]
____________________________________________________________________________________________________
scale4b3_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b3_branch2c[0][0]
____________________________________________________________________________________________________
res4b3 (Add)                    (None, 24, 24, 1024) 0           scale4b3_branch2c[0][0]
                                                                 res4b2_relu[0][0]
____________________________________________________________________________________________________
res4b3_relu (Activation)        (None, 24, 24, 1024) 0           res4b3[0][0]
____________________________________________________________________________________________________
res4b4_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b3_relu[0][0]
____________________________________________________________________________________________________
bn4b4_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b4_branch2a[0][0]
____________________________________________________________________________________________________
scale4b4_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b4_branch2a[0][0]
____________________________________________________________________________________________________
res4b4_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b4_branch2a[0][0]
____________________________________________________________________________________________________
res4b4_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b4_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b4_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b4_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b4_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b4_branch2b[0][0]
____________________________________________________________________________________________________
```

```
scale4b4_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b4_branch2b[0][0]
________________________________________________________________________________________________
res4b4_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b4_branch2b[0][0]
________________________________________________________________________________________________
res4b4_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b4_branch2b_relu[0][0]
________________________________________________________________________________________________
bn4b4_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b4_branch2c[0][0]
________________________________________________________________________________________________
scale4b4_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b4_branch2c[0][0]
________________________________________________________________________________________________
res4b4 (Add)                    (None, 24, 24, 1024) 0           scale4b4_branch2c[0][0]
                                                                 res4b3_relu[0][0]
________________________________________________________________________________________________
res4b4_relu (Activation)        (None, 24, 24, 1024) 0           res4b4[0][0]
________________________________________________________________________________________________
res4b5_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b4_relu[0][0]
________________________________________________________________________________________________
bn4b5_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b5_branch2a[0][0]
________________________________________________________________________________________________
scale4b5_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b5_branch2a[0][0]
________________________________________________________________________________________________
res4b5_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b5_branch2a[0][0]
________________________________________________________________________________________________
res4b5_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b5_branch2a_relu[0][0]
________________________________________________________________________________________________
res4b5_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b5_branch2b_zeropadding[0][0]
________________________________________________________________________________________________
bn4b5_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b5_branch2b[0][0]
________________________________________________________________________________________________
scale4b5_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b5_branch2b[0][0]
________________________________________________________________________________________________
res4b5_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b5_branch2b[0][0]
________________________________________________________________________________________________
res4b5_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b5_branch2b_relu[0][0]
________________________________________________________________________________________________
bn4b5_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b5_branch2c[0][0]
________________________________________________________________________________________________
scale4b5_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b5_branch2c[0][0]
________________________________________________________________________________________________
res4b5 (Add)                    (None, 24, 24, 1024) 0           scale4b5_branch2c[0][0]
                                                                 res4b4_relu[0][0]
________________________________________________________________________________________________
res4b5_relu (Activation)        (None, 24, 24, 1024) 0           res4b5[0][0]
________________________________________________________________________________________________
res4b6_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b5_relu[0][0]
________________________________________________________________________________________________
bn4b6_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b6_branch2a[0][0]
____________________________________________________________________________________________________
scale4b6_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b6_branch2a[0][0]
____________________________________________________________________________________________________
res4b6_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b6_branch2a[0][0]
____________________________________________________________________________________________________
res4b6_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b6_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b6_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b6_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b6_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b6_branch2b[0][0]
____________________________________________________________________________________________________
scale4b6_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b6_branch2b[0][0]
____________________________________________________________________________________________________
res4b6_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b6_branch2b[0][0]
____________________________________________________________________________________________________
res4b6_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b6_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b6_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b6_branch2c[0][0]
____________________________________________________________________________________________________
scale4b6_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b6_branch2c[0][0]
____________________________________________________________________________________________________
res4b6 (Add)                    (None, 24, 24, 1024) 0           scale4b6_branch2c[0][0]
                                                                 res4b5_relu[0][0]
____________________________________________________________________________________________________
res4b6_relu (Activation)        (None, 24, 24, 1024) 0           res4b6[0][0]
____________________________________________________________________________________________________
res4b7_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b6_relu[0][0]
____________________________________________________________________________________________________
bn4b7_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b7_branch2a[0][0]
____________________________________________________________________________________________________
scale4b7_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b7_branch2a[0][0]
____________________________________________________________________________________________________
res4b7_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b7_branch2a[0][0]
____________________________________________________________________________________________________
res4b7_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b7_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b7_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b7_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b7_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b7_branch2b[0][0]
____________________________________________________________________________________________________
scale4b7_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b7_branch2b[0][0]
____________________________________________________________________________________________________
res4b7_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b7_branch2b[0][0]
____________________________________________________________________________________________________
res4b7_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b7_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b7_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b7_branch2c[0][0]
____________________________________________________________________________________________________
scale4b7_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b7_branch2c[0][0]
____________________________________________________________________________________________________
res4b7 (Add)                    (None, 24, 24, 1024) 0           scale4b7_branch2c[0][0]
                                                                 res4b6_relu[0][0]
____________________________________________________________________________________________________
res4b7_relu (Activation)        (None, 24, 24, 1024) 0           res4b7[0][0]
____________________________________________________________________________________________________
res4b8_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b7_relu[0][0]
____________________________________________________________________________________________________
bn4b8_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b8_branch2a[0][0]
____________________________________________________________________________________________________
scale4b8_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b8_branch2a[0][0]
____________________________________________________________________________________________________
res4b8_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b8_branch2a[0][0]
____________________________________________________________________________________________________
res4b8_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b8_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b8_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b8_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b8_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b8_branch2b[0][0]
____________________________________________________________________________________________________
scale4b8_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b8_branch2b[0][0]
____________________________________________________________________________________________________
res4b8_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b8_branch2b[0][0]
____________________________________________________________________________________________________
res4b8_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b8_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b8_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b8_branch2c[0][0]
____________________________________________________________________________________________________
scale4b8_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b8_branch2c[0][0]
____________________________________________________________________________________________________
res4b8 (Add)                    (None, 24, 24, 1024) 0           scale4b8_branch2c[0][0]
                                                                 res4b7_relu[0][0]
____________________________________________________________________________________________________
res4b8_relu (Activation)        (None, 24, 24, 1024) 0           res4b8[0][0]
____________________________________________________________________________________________________
res4b9_branch2a (Conv2D)        (None, 24, 24, 256)  262144      res4b8_relu[0][0]
____________________________________________________________________________________________________
bn4b9_branch2a (BatchNormalizat (None, 24, 24, 256)  1024        res4b9_branch2a[0][0]
____________________________________________________________________________________________________
scale4b9_branch2a (Scale)       (None, 24, 24, 256)  512         bn4b9_branch2a[0][0]
____________________________________________________________________________________________________
res4b9_branch2a_relu (Activatio (None, 24, 24, 256)  0           scale4b9_branch2a[0][0]
____________________________________________________________________________________________________
res4b9_branch2b_zeropadding (Ze (None, 26, 26, 256)  0           res4b9_branch2a_relu[0][0]
____________________________________________________________________________________________________
```

```
res4b9_branch2b (Conv2D)        (None, 24, 24, 256)  589824      res4b9_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn4b9_branch2b (BatchNormalizat (None, 24, 24, 256)  1024        res4b9_branch2b[0][0]
____________________________________________________________________________________________________
scale4b9_branch2b (Scale)       (None, 24, 24, 256)  512         bn4b9_branch2b[0][0]
____________________________________________________________________________________________________
res4b9_branch2b_relu (Activatio (None, 24, 24, 256)  0           scale4b9_branch2b[0][0]
____________________________________________________________________________________________________
res4b9_branch2c (Conv2D)        (None, 24, 24, 1024) 262144      res4b9_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b9_branch2c (BatchNormalizat (None, 24, 24, 1024) 4096        res4b9_branch2c[0][0]
____________________________________________________________________________________________________
scale4b9_branch2c (Scale)       (None, 24, 24, 1024) 2048        bn4b9_branch2c[0][0]
____________________________________________________________________________________________________
res4b9 (Add)                    (None, 24, 24, 1024) 0           scale4b9_branch2c[0][0]
                                                                 res4b8_relu[0][0]
____________________________________________________________________________________________________
res4b9_relu (Activation)        (None, 24, 24, 1024) 0           res4b9[0][0]
____________________________________________________________________________________________________
res4b10_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b9_relu[0][0]
____________________________________________________________________________________________________
bn4b10_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b10_branch2a[0][0]
____________________________________________________________________________________________________
scale4b10_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b10_branch2a[0][0]
____________________________________________________________________________________________________
res4b10_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b10_branch2a[0][0]
____________________________________________________________________________________________________
res4b10_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b10_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b10_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b10_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b10_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b10_branch2b[0][0]
____________________________________________________________________________________________________
scale4b10_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b10_branch2b[0][0]
____________________________________________________________________________________________________
res4b10_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b10_branch2b[0][0]
____________________________________________________________________________________________________
res4b10_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b10_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b10_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b10_branch2c[0][0]
____________________________________________________________________________________________________
scale4b10_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b10_branch2c[0][0]
____________________________________________________________________________________________________
res4b10 (Add)                   (None, 24, 24, 1024) 0           scale4b10_branch2c[0][0]
                                                                 res4b9_relu[0][0]
____________________________________________________________________________________________________
res4b10_relu (Activation)       (None, 24, 24, 1024) 0           res4b10[0][0]
```

```
____________________________________________________________________________________________________
res4b11_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b10_relu[0][0]
____________________________________________________________________________________________________
bn4b11_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b11_branch2a[0][0]
____________________________________________________________________________________________________
scale4b11_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b11_branch2a[0][0]
____________________________________________________________________________________________________
res4b11_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b11_branch2a[0][0]
____________________________________________________________________________________________________
res4b11_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b11_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b11_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b11_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b11_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b11_branch2b[0][0]
____________________________________________________________________________________________________
scale4b11_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b11_branch2b[0][0]
____________________________________________________________________________________________________
res4b11_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b11_branch2b[0][0]
____________________________________________________________________________________________________
res4b11_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b11_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b11_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b11_branch2c[0][0]
____________________________________________________________________________________________________
scale4b11_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b11_branch2c[0][0]
____________________________________________________________________________________________________
res4b11 (Add)                   (None, 24, 24, 1024) 0           scale4b11_branch2c[0][0]
                                                                 res4b10_relu[0][0]
____________________________________________________________________________________________________
res4b11_relu (Activation)       (None, 24, 24, 1024) 0           res4b11[0][0]
____________________________________________________________________________________________________
res4b12_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b11_relu[0][0]
____________________________________________________________________________________________________
bn4b12_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b12_branch2a[0][0]
____________________________________________________________________________________________________
scale4b12_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b12_branch2a[0][0]
____________________________________________________________________________________________________
res4b12_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b12_branch2a[0][0]
____________________________________________________________________________________________________
res4b12_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b12_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b12_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b12_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b12_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b12_branch2b[0][0]
____________________________________________________________________________________________________
scale4b12_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b12_branch2b[0][0]
____________________________________________________________________________________________________
res4b12_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b12_branch2b[0][0]
____________________________________________________________________________________________________
res4b12_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b12_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b12_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b12_branch2c[0][0]
____________________________________________________________________________________________________
scale4b12_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b12_branch2c[0][0]
____________________________________________________________________________________________________
res4b12 (Add)                   (None, 24, 24, 1024) 0           scale4b12_branch2c[0][0]
                                                                 res4b11_relu[0][0]
____________________________________________________________________________________________________
res4b12_relu (Activation)       (None, 24, 24, 1024) 0           res4b12[0][0]
____________________________________________________________________________________________________
res4b13_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b12_relu[0][0]
____________________________________________________________________________________________________
bn4b13_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b13_branch2a[0][0]
____________________________________________________________________________________________________
scale4b13_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b13_branch2a[0][0]
____________________________________________________________________________________________________
res4b13_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b13_branch2a[0][0]
____________________________________________________________________________________________________
res4b13_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b13_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b13_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b13_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b13_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b13_branch2b[0][0]
____________________________________________________________________________________________________
scale4b13_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b13_branch2b[0][0]
____________________________________________________________________________________________________
res4b13_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b13_branch2b[0][0]
____________________________________________________________________________________________________
res4b13_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b13_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b13_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b13_branch2c[0][0]
____________________________________________________________________________________________________
scale4b13_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b13_branch2c[0][0]
____________________________________________________________________________________________________
res4b13 (Add)                   (None, 24, 24, 1024) 0           scale4b13_branch2c[0][0]
                                                                 res4b12_relu[0][0]
____________________________________________________________________________________________________
res4b13_relu (Activation)       (None, 24, 24, 1024) 0           res4b13[0][0]
____________________________________________________________________________________________________
res4b14_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b13_relu[0][0]
____________________________________________________________________________________________________
bn4b14_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b14_branch2a[0][0]
____________________________________________________________________________________________________
scale4b14_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b14_branch2a[0][0]
____________________________________________________________________________________________________
```

```
res4b14_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b14_branch2a[0][0]
____________________________________________________________________________________________________
res4b14_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b14_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b14_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b14_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b14_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b14_branch2b[0][0]
____________________________________________________________________________________________________
scale4b14_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b14_branch2b[0][0]
____________________________________________________________________________________________________
res4b14_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b14_branch2b[0][0]
____________________________________________________________________________________________________
res4b14_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b14_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b14_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b14_branch2c[0][0]
____________________________________________________________________________________________________
scale4b14_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b14_branch2c[0][0]
____________________________________________________________________________________________________
res4b14 (Add)                   (None, 24, 24, 1024) 0           scale4b14_branch2c[0][0]
                                                                 res4b13_relu[0][0]
____________________________________________________________________________________________________
res4b14_relu (Activation)       (None, 24, 24, 1024) 0           res4b14[0][0]
____________________________________________________________________________________________________
res4b15_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b14_relu[0][0]
____________________________________________________________________________________________________
bn4b15_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b15_branch2a[0][0]
____________________________________________________________________________________________________
scale4b15_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b15_branch2a[0][0]
____________________________________________________________________________________________________
res4b15_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b15_branch2a[0][0]
____________________________________________________________________________________________________
res4b15_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b15_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b15_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b15_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b15_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b15_branch2b[0][0]
____________________________________________________________________________________________________
scale4b15_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b15_branch2b[0][0]
____________________________________________________________________________________________________
res4b15_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b15_branch2b[0][0]
____________________________________________________________________________________________________
res4b15_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b15_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b15_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b15_branch2c[0][0]
____________________________________________________________________________________________________
scale4b15_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b15_branch2c[0][0]
____________________________________________________________________________________________________
```

```
res4b15 (Add)                   (None, 24, 24, 1024) 0           scale4b15_branch2c[0][0]
                                                                 res4b14_relu[0][0]
__________________________________________________________________________________________________
res4b15_relu (Activation)       (None, 24, 24, 1024) 0           res4b15[0][0]
__________________________________________________________________________________________________
res4b16_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b15_relu[0][0]
__________________________________________________________________________________________________
bn4b16_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b16_branch2a[0][0]
__________________________________________________________________________________________________
scale4b16_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b16_branch2a[0][0]
__________________________________________________________________________________________________
res4b16_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b16_branch2a[0][0]
__________________________________________________________________________________________________
res4b16_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b16_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b16_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b16_branch2b_zeropadding[0][0
__________________________________________________________________________________________________
bn4b16_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b16_branch2b[0][0]
__________________________________________________________________________________________________
scale4b16_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b16_branch2b[0][0]
__________________________________________________________________________________________________
res4b16_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b16_branch2b[0][0]
__________________________________________________________________________________________________
res4b16_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b16_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn4b16_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b16_branch2c[0][0]
__________________________________________________________________________________________________
scale4b16_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b16_branch2c[0][0]
__________________________________________________________________________________________________
res4b16 (Add)                   (None, 24, 24, 1024) 0           scale4b16_branch2c[0][0]
                                                                 res4b15_relu[0][0]
__________________________________________________________________________________________________
res4b16_relu (Activation)       (None, 24, 24, 1024) 0           res4b16[0][0]
__________________________________________________________________________________________________
res4b17_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b16_relu[0][0]
__________________________________________________________________________________________________
bn4b17_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b17_branch2a[0][0]
__________________________________________________________________________________________________
scale4b17_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b17_branch2a[0][0]
__________________________________________________________________________________________________
res4b17_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b17_branch2a[0][0]
__________________________________________________________________________________________________
res4b17_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b17_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b17_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b17_branch2b_zeropadding[0][0
__________________________________________________________________________________________________
bn4b17_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b17_branch2b[0][0]
____________________________________________________________________________________________________
scale4b17_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b17_branch2b[0][0]
____________________________________________________________________________________________________
res4b17_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b17_branch2b[0][0]
____________________________________________________________________________________________________
res4b17_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b17_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b17_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b17_branch2c[0][0]
____________________________________________________________________________________________________
scale4b17_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b17_branch2c[0][0]
____________________________________________________________________________________________________
res4b17 (Add)                   (None, 24, 24, 1024) 0           scale4b17_branch2c[0][0]
                                                                 res4b16_relu[0][0]
____________________________________________________________________________________________________
res4b17_relu (Activation)       (None, 24, 24, 1024) 0           res4b17[0][0]
____________________________________________________________________________________________________
res4b18_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b17_relu[0][0]
____________________________________________________________________________________________________
bn4b18_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b18_branch2a[0][0]
____________________________________________________________________________________________________
scale4b18_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b18_branch2a[0][0]
____________________________________________________________________________________________________
res4b18_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b18_branch2a[0][0]
____________________________________________________________________________________________________
res4b18_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b18_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b18_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b18_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b18_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b18_branch2b[0][0]
____________________________________________________________________________________________________
scale4b18_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b18_branch2b[0][0]
____________________________________________________________________________________________________
res4b18_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b18_branch2b[0][0]
____________________________________________________________________________________________________
res4b18_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b18_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b18_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b18_branch2c[0][0]
____________________________________________________________________________________________________
scale4b18_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b18_branch2c[0][0]
____________________________________________________________________________________________________
res4b18 (Add)                   (None, 24, 24, 1024) 0           scale4b18_branch2c[0][0]
                                                                 res4b17_relu[0][0]
____________________________________________________________________________________________________
res4b18_relu (Activation)       (None, 24, 24, 1024) 0           res4b18[0][0]
____________________________________________________________________________________________________
res4b19_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b18_relu[0][0]
____________________________________________________________________________________________________
```

```
bn4b19_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b19_branch2a[0][0]
__________________________________________________________________________________________________
scale4b19_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b19_branch2a[0][0]
__________________________________________________________________________________________________
res4b19_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b19_branch2a[0][0]
__________________________________________________________________________________________________
res4b19_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b19_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b19_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b19_branch2b_zeropadding[0][0
__________________________________________________________________________________________________
bn4b19_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b19_branch2b[0][0]
__________________________________________________________________________________________________
scale4b19_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b19_branch2b[0][0]
__________________________________________________________________________________________________
res4b19_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b19_branch2b[0][0]
__________________________________________________________________________________________________
res4b19_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b19_branch2b_relu[0][0]
__________________________________________________________________________________________________
bn4b19_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b19_branch2c[0][0]
__________________________________________________________________________________________________
scale4b19_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b19_branch2c[0][0]
__________________________________________________________________________________________________
res4b19 (Add)                   (None, 24, 24, 1024) 0           scale4b19_branch2c[0][0]
                                                                 res4b18_relu[0][0]
__________________________________________________________________________________________________
res4b19_relu (Activation)       (None, 24, 24, 1024) 0           res4b19[0][0]
__________________________________________________________________________________________________
res4b20_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b19_relu[0][0]
__________________________________________________________________________________________________
bn4b20_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b20_branch2a[0][0]
__________________________________________________________________________________________________
scale4b20_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b20_branch2a[0][0]
__________________________________________________________________________________________________
res4b20_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b20_branch2a[0][0]
__________________________________________________________________________________________________
res4b20_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b20_branch2a_relu[0][0]
__________________________________________________________________________________________________
res4b20_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b20_branch2b_zeropadding[0][0
__________________________________________________________________________________________________
bn4b20_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b20_branch2b[0][0]
__________________________________________________________________________________________________
scale4b20_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b20_branch2b[0][0]
__________________________________________________________________________________________________
res4b20_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b20_branch2b[0][0]
__________________________________________________________________________________________________
res4b20_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b20_branch2b_relu[0][0]
__________________________________________________________________________________________________
```

```
bn4b20_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b20_branch2c[0][0]
____________________________________________________________________________________________________
scale4b20_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b20_branch2c[0][0]
____________________________________________________________________________________________________
res4b20 (Add)                   (None, 24, 24, 1024) 0           scale4b20_branch2c[0][0]
                                                                 res4b19_relu[0][0]
____________________________________________________________________________________________________
res4b20_relu (Activation)       (None, 24, 24, 1024) 0           res4b20[0][0]
____________________________________________________________________________________________________
res4b21_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b20_relu[0][0]
____________________________________________________________________________________________________
bn4b21_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b21_branch2a[0][0]
____________________________________________________________________________________________________
scale4b21_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b21_branch2a[0][0]
____________________________________________________________________________________________________
res4b21_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b21_branch2a[0][0]
____________________________________________________________________________________________________
res4b21_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b21_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b21_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b21_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b21_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b21_branch2b[0][0]
____________________________________________________________________________________________________
scale4b21_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b21_branch2b[0][0]
____________________________________________________________________________________________________
res4b21_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b21_branch2b[0][0]
____________________________________________________________________________________________________
res4b21_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b21_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b21_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b21_branch2c[0][0]
____________________________________________________________________________________________________
scale4b21_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b21_branch2c[0][0]
____________________________________________________________________________________________________
res4b21 (Add)                   (None, 24, 24, 1024) 0           scale4b21_branch2c[0][0]
                                                                 res4b20_relu[0][0]
____________________________________________________________________________________________________
res4b21_relu (Activation)       (None, 24, 24, 1024) 0           res4b21[0][0]
____________________________________________________________________________________________________
res4b22_branch2a (Conv2D)       (None, 24, 24, 256)  262144      res4b21_relu[0][0]
____________________________________________________________________________________________________
bn4b22_branch2a (BatchNormaliza (None, 24, 24, 256)  1024        res4b22_branch2a[0][0]
____________________________________________________________________________________________________
scale4b22_branch2a (Scale)      (None, 24, 24, 256)  512         bn4b22_branch2a[0][0]
____________________________________________________________________________________________________
res4b22_branch2a_relu (Activati (None, 24, 24, 256)  0           scale4b22_branch2a[0][0]
____________________________________________________________________________________________________
res4b22_branch2b_zeropadding (Z (None, 26, 26, 256)  0           res4b22_branch2a_relu[0][0]
____________________________________________________________________________________________________
res4b22_branch2b (Conv2D)       (None, 24, 24, 256)  589824      res4b22_branch2b_zeropadding[0][0
____________________________________________________________________________________________________
bn4b22_branch2b (BatchNormaliza (None, 24, 24, 256)  1024        res4b22_branch2b[0][0]
____________________________________________________________________________________________________
scale4b22_branch2b (Scale)      (None, 24, 24, 256)  512         bn4b22_branch2b[0][0]
____________________________________________________________________________________________________
res4b22_branch2b_relu (Activati (None, 24, 24, 256)  0           scale4b22_branch2b[0][0]
____________________________________________________________________________________________________
res4b22_branch2c (Conv2D)       (None, 24, 24, 1024) 262144      res4b22_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn4b22_branch2c (BatchNormaliza (None, 24, 24, 1024) 4096        res4b22_branch2c[0][0]
____________________________________________________________________________________________________
scale4b22_branch2c (Scale)      (None, 24, 24, 1024) 2048        bn4b22_branch2c[0][0]
____________________________________________________________________________________________________
res4b22 (Add)                   (None, 24, 24, 1024) 0           scale4b22_branch2c[0][0]
                                                                 res4b21_relu[0][0]
____________________________________________________________________________________________________
res4b22_relu (Activation)       (None, 24, 24, 1024) 0           res4b22[0][0]
____________________________________________________________________________________________________
res5a_branch2a (Conv2D)         (None, 12, 12, 512)  524288      res4b22_relu[0][0]
____________________________________________________________________________________________________
bn5a_branch2a (BatchNormalizati (None, 12, 12, 512)  2048        res5a_branch2a[0][0]
____________________________________________________________________________________________________
scale5a_branch2a (Scale)        (None, 12, 12, 512)  1024        bn5a_branch2a[0][0]
____________________________________________________________________________________________________
res5a_branch2a_relu (Activation (None, 12, 12, 512)  0           scale5a_branch2a[0][0]
____________________________________________________________________________________________________
res5a_branch2b_zeropadding (Zer (None, 14, 14, 512)  0           res5a_branch2a_relu[0][0]
____________________________________________________________________________________________________
res5a_branch2b (Conv2D)         (None, 12, 12, 512)  2359296     res5a_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn5a_branch2b (BatchNormalizati (None, 12, 12, 512)  2048        res5a_branch2b[0][0]
____________________________________________________________________________________________________
scale5a_branch2b (Scale)        (None, 12, 12, 512)  1024        bn5a_branch2b[0][0]
____________________________________________________________________________________________________
res5a_branch2b_relu (Activation (None, 12, 12, 512)  0           scale5a_branch2b[0][0]
____________________________________________________________________________________________________
res5a_branch2c (Conv2D)         (None, 12, 12, 2048) 1048576     res5a_branch2b_relu[0][0]
____________________________________________________________________________________________________
res5a_branch1 (Conv2D)          (None, 12, 12, 2048) 2097152     res4b22_relu[0][0]
____________________________________________________________________________________________________
bn5a_branch2c (BatchNormalizati (None, 12, 12, 2048) 8192        res5a_branch2c[0][0]
____________________________________________________________________________________________________
bn5a_branch1 (BatchNormalizatio (None, 12, 12, 2048) 8192        res5a_branch1[0][0]
____________________________________________________________________________________________________
scale5a_branch2c (Scale)        (None, 12, 12, 2048) 4096        bn5a_branch2c[0][0]
```

```
--------------------------------------------------------------------------------------------------
scale5a_branch1 (Scale)         (None, 12, 12, 2048) 4096        bn5a_branch1[0][0]

--------------------------------------------------------------------------------------------------
res5a (Add)                     (None, 12, 12, 2048) 0           scale5a_branch2c[0][0]
                                                                 scale5a_branch1[0][0]

--------------------------------------------------------------------------------------------------
res5a_relu (Activation)         (None, 12, 12, 2048) 0           res5a[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2a (Conv2D)         (None, 12, 12, 512)  1048576     res5a_relu[0][0]

--------------------------------------------------------------------------------------------------
bn5b_branch2a (BatchNormalizati (None, 12, 12, 512)  2048        res5b_branch2a[0][0]

--------------------------------------------------------------------------------------------------
scale5b_branch2a (Scale)        (None, 12, 12, 512)  1024        bn5b_branch2a[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2a_relu (Activation (None, 12, 12, 512)  0           scale5b_branch2a[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2b_zeropadding (Zer (None, 14, 14, 512)  0           res5b_branch2a_relu[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2b (Conv2D)         (None, 12, 12, 512)  2359296     res5b_branch2b_zeropadding[0][0]

--------------------------------------------------------------------------------------------------
bn5b_branch2b (BatchNormalizati (None, 12, 12, 512)  2048        res5b_branch2b[0][0]

--------------------------------------------------------------------------------------------------
scale5b_branch2b (Scale)        (None, 12, 12, 512)  1024        bn5b_branch2b[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2b_relu (Activation (None, 12, 12, 512)  0           scale5b_branch2b[0][0]

--------------------------------------------------------------------------------------------------
res5b_branch2c (Conv2D)         (None, 12, 12, 2048) 1048576     res5b_branch2b_relu[0][0]

--------------------------------------------------------------------------------------------------
bn5b_branch2c (BatchNormalizati (None, 12, 12, 2048) 8192        res5b_branch2c[0][0]

--------------------------------------------------------------------------------------------------
scale5b_branch2c (Scale)        (None, 12, 12, 2048) 4096        bn5b_branch2c[0][0]

--------------------------------------------------------------------------------------------------
res5b (Add)                     (None, 12, 12, 2048) 0           scale5b_branch2c[0][0]
                                                                 res5a_relu[0][0]

--------------------------------------------------------------------------------------------------
res5b_relu (Activation)         (None, 12, 12, 2048) 0           res5b[0][0]

--------------------------------------------------------------------------------------------------
res5c_branch2a (Conv2D)         (None, 12, 12, 512)  1048576     res5b_relu[0][0]

--------------------------------------------------------------------------------------------------
bn5c_branch2a (BatchNormalizati (None, 12, 12, 512)  2048        res5c_branch2a[0][0]

--------------------------------------------------------------------------------------------------
scale5c_branch2a (Scale)        (None, 12, 12, 512)  1024        bn5c_branch2a[0][0]

--------------------------------------------------------------------------------------------------
res5c_branch2a_relu (Activation (None, 12, 12, 512)  0           scale5c_branch2a[0][0]

--------------------------------------------------------------------------------------------------
res5c_branch2b_zeropadding (Zer (None, 14, 14, 512)  0           res5c_branch2a_relu[0][0]

--------------------------------------------------------------------------------------------------
```

```
res5c_branch2b (Conv2D)         (None, 12, 12, 512)  2359296     res5c_branch2b_zeropadding[0][0]
____________________________________________________________________________________________________
bn5c_branch2b (BatchNormalizati (None, 12, 12, 512)  2048        res5c_branch2b[0][0]
____________________________________________________________________________________________________
scale5c_branch2b (Scale)        (None, 12, 12, 512)  1024        bn5c_branch2b[0][0]
____________________________________________________________________________________________________
res5c_branch2b_relu (Activation (None, 12, 12, 512)  0           scale5c_branch2b[0][0]
____________________________________________________________________________________________________
res5c_branch2c (Conv2D)         (None, 12, 12, 2048) 1048576     res5c_branch2b_relu[0][0]
____________________________________________________________________________________________________
bn5c_branch2c (BatchNormalizati (None, 12, 12, 2048) 8192        res5c_branch2c[0][0]
____________________________________________________________________________________________________
scale5c_branch2c (Scale)        (None, 12, 12, 2048) 4096        bn5c_branch2c[0][0]
____________________________________________________________________________________________________
res5c (Add)                     (None, 12, 12, 2048) 0           scale5c_branch2c[0][0]
                                                                 res5b_relu[0][0]
____________________________________________________________________________________________________
res5c_relu (Activation)         (None, 12, 12, 2048) 0           res5c[0][0]
____________________________________________________________________________________________________
resnet_map1 (Conv2D)            (None, 12, 12, 512)  1049088     res5c_relu[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h1_relu1 (ReLU)        (None, 12, 12, 512)  0           resnet_map1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h1_conv1 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_h1_relu1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h1_relu2 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_h1_conv1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h1_conv2 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_h1_relu2[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h1_sum (Add)           (None, 12, 12, 512)  0           rb_4_rcu_h1_conv2[0][0]
                                                                 resnet_map1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h2_relu1 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h2_conv1 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_h2_relu1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h2_relu2 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_h2_conv1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h2_conv2 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_h2_relu2[0][0]
____________________________________________________________________________________________________
rb_4_rcu_h2_sum (Add)           (None, 12, 12, 512)  0           rb_4_rcu_h2_conv2[0][0]
                                                                 rb_4_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
rb_4_rcu_o1_relu1 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_h2_sum[0][0]
____________________________________________________________________________________________________
rb_4_rcu_o1_conv1 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_o1_relu1[0][0]
____________________________________________________________________________________________________
```

```
rb_4_rcu_o1_relu2 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_o1_conv1[0][0]
____________________________________________________________________________________________________
rb_4_rcu_o1_conv2 (Conv2D)      (None, 12, 12, 512)  2359808     rb_4_rcu_o1_relu2[0][0]
____________________________________________________________________________________________________
rb_4_rcu_o1_sum (Add)           (None, 12, 12, 512)  0           rb_4_rcu_o1_conv2[0][0]
                                                                 rb_4_rcu_h2_sum[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l1_relu1 (ReLU)        (None, 12, 12, 512)  0           rb_4_rcu_o1_sum[0][0]
____________________________________________________________________________________________________
resnet_map2 (Conv2D)            (None, 24, 24, 256)  262400      res4b22_relu[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l1_conv1 (Conv2D)      (None, 12, 12, 512)  2359808     rb_3_rcu_l1_relu1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h1_relu1 (ReLU)        (None, 24, 24, 256)  0           resnet_map2[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l1_relu2 (ReLU)        (None, 12, 12, 512)  0           rb_3_rcu_l1_conv1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h1_conv1 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_h1_relu1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l1_conv2 (Conv2D)      (None, 12, 12, 512)  2359808     rb_3_rcu_l1_relu2[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h1_relu2 (ReLU)        (None, 24, 24, 256)  0           rb_3_rcu_h1_conv1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l1_sum (Add)           (None, 12, 12, 512)  0           rb_3_rcu_l1_conv2[0][0]
                                                                 rb_4_rcu_o1_sum[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h1_conv2 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_h1_relu2[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l2_relu1 (ReLU)        (None, 12, 12, 512)  0           rb_3_rcu_l1_sum[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h1_sum (Add)           (None, 24, 24, 256)  0           rb_3_rcu_h1_conv2[0][0]
                                                                 resnet_map2[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l2_conv1 (Conv2D)      (None, 12, 12, 512)  2359808     rb_3_rcu_l2_relu1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h2_relu1 (ReLU)        (None, 24, 24, 256)  0           rb_3_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l2_relu2 (ReLU)        (None, 12, 12, 512)  0           rb_3_rcu_l2_conv1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h2_conv1 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_h2_relu1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l2_conv2 (Conv2D)      (None, 12, 12, 512)  2359808     rb_3_rcu_l2_relu2[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h2_relu2 (ReLU)        (None, 24, 24, 256)  0           rb_3_rcu_h2_conv1[0][0]
____________________________________________________________________________________________________
rb_3_rcu_l2_sum (Add)           (None, 12, 12, 512)  0           rb_3_rcu_l2_conv2[0][0]
                                                                 rb_3_rcu_l1_sum[0][0]
```

```
____________________________________________________________________________________________________
rb_3_rcu_h2_conv2 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_h2_relu2[0][0]
____________________________________________________________________________________________________
rb_3_mrf_conv_lo (Conv2D)       (None, 12, 12, 256)  1179904     rb_3_rcu_l2_sum[0][0]
____________________________________________________________________________________________________
rb_3_rcu_h2_sum (Add)           (None, 24, 24, 256)  0           rb_3_rcu_h2_conv2[0][0]
                                                                 rb_3_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 12, 12, 256)  1024        rb_3_mrf_conv_lo[0][0]
____________________________________________________________________________________________________
rb_3_mrf_conv_hi (Conv2D)       (None, 24, 24, 256)  590080      rb_3_rcu_h2_sum[0][0]
____________________________________________________________________________________________________
rb_3_mrf_up (UpSampling2D)      (None, 24, 24, 256)  0           batch_normalization_9[0][0]
____________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 24, 24, 256)  1024        rb_3_mrf_conv_hi[0][0]
____________________________________________________________________________________________________
rb_3_mrf_sum (Add)              (None, 24, 24, 256)  0           rb_3_mrf_up[0][0]
                                                                 batch_normalization_10[0][0]
____________________________________________________________________________________________________
rb_3_crp_relu (ReLU)            (None, 24, 24, 256)  0           rb_3_mrf_sum[0][0]
____________________________________________________________________________________________________
rb_3_crp_conv1 (Conv2D)         (None, 24, 24, 256)  590080      rb_3_crp_relu[0][0]
____________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 24, 24, 256)  1024        rb_3_crp_conv1[0][0]
____________________________________________________________________________________________________
rb_3_crp_pool1 (MaxPooling2D)   (None, 24, 24, 256)  0           batch_normalization_11[0][0]
____________________________________________________________________________________________________
rb_3_crp_conv2 (Conv2D)         (None, 24, 24, 256)  590080      rb_3_crp_pool1[0][0]
____________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 24, 24, 256)  1024        rb_3_crp_conv2[0][0]
____________________________________________________________________________________________________
rb_3_crp_pool2 (MaxPooling2D)   (None, 24, 24, 256)  0           batch_normalization_12[0][0]
____________________________________________________________________________________________________
rb_3_crp_conv3 (Conv2D)         (None, 24, 24, 256)  590080      rb_3_crp_pool2[0][0]
____________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 24, 24, 256)  1024        rb_3_crp_conv3[0][0]
____________________________________________________________________________________________________
rb_3_crp_pool3 (MaxPooling2D)   (None, 24, 24, 256)  0           batch_normalization_13[0][0]
____________________________________________________________________________________________________
rb_3_crp_conv4 (Conv2D)         (None, 24, 24, 256)  590080      rb_3_crp_pool3[0][0]
____________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 24, 24, 256)  1024        rb_3_crp_conv4[0][0]
____________________________________________________________________________________________________
rb_3_crp_pool4 (MaxPooling2D)   (None, 24, 24, 256)  0           batch_normalization_14[0][0]
____________________________________________________________________________________________________
rb_3_crp_sum (Add)              (None, 24, 24, 256)  0           rb_3_crp_relu[0][0]
                                                                 rb_3_crp_pool1[0][0]
```

```
                                                                rb_3_crp_pool2[0][0]
                                                                rb_3_crp_pool3[0][0]
                                                                rb_3_crp_pool4[0][0]
__________________________________________________________________________________________________
rb_3_rcu_o1_relu1 (ReLU)        (None, 24, 24, 256)  0           rb_3_crp_sum[0][0]
__________________________________________________________________________________________________
rb_3_rcu_o1_conv1 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_o1_relu1[0][0]
__________________________________________________________________________________________________
rb_3_rcu_o1_relu2 (ReLU)        (None, 24, 24, 256)  0           rb_3_rcu_o1_conv1[0][0]
__________________________________________________________________________________________________
rb_3_rcu_o1_conv2 (Conv2D)      (None, 24, 24, 256)  590080      rb_3_rcu_o1_relu2[0][0]
__________________________________________________________________________________________________
rb_3_rcu_o1_sum (Add)           (None, 24, 24, 256)  0           rb_3_rcu_o1_conv2[0][0]
                                                                 rb_3_crp_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l1_relu1 (ReLU)        (None, 24, 24, 256)  0           rb_3_rcu_o1_sum[0][0]
__________________________________________________________________________________________________
resnet_map3 (Conv2D)            (None, 48, 48, 256)  131328      res3b3_relu[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l1_conv1 (Conv2D)      (None, 24, 24, 256)  590080      rb_2_rcu_l1_relu1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h1_relu1 (ReLU)        (None, 48, 48, 256)  0           resnet_map3[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l1_relu2 (ReLU)        (None, 24, 24, 256)  0           rb_2_rcu_l1_conv1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h1_conv1 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_h1_relu1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l1_conv2 (Conv2D)      (None, 24, 24, 256)  590080      rb_2_rcu_l1_relu2[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h1_relu2 (ReLU)        (None, 48, 48, 256)  0           rb_2_rcu_h1_conv1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l1_sum (Add)           (None, 24, 24, 256)  0           rb_2_rcu_l1_conv2[0][0]
                                                                 rb_3_rcu_o1_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h1_conv2 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_h1_relu2[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l2_relu1 (ReLU)        (None, 24, 24, 256)  0           rb_2_rcu_l1_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h1_sum (Add)           (None, 48, 48, 256)  0           rb_2_rcu_h1_conv2[0][0]
                                                                 resnet_map3[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l2_conv1 (Conv2D)      (None, 24, 24, 256)  590080      rb_2_rcu_l2_relu1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h2_relu1 (ReLU)        (None, 48, 48, 256)  0           rb_2_rcu_h1_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l2_relu2 (ReLU)        (None, 24, 24, 256)  0           rb_2_rcu_l2_conv1[0][0]
__________________________________________________________________________________________________
```

```
rb_2_rcu_h2_conv1 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_h2_relu1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l2_conv2 (Conv2D)      (None, 24, 24, 256)  590080      rb_2_rcu_l2_relu2[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h2_relu2 (ReLU)        (None, 48, 48, 256)  0           rb_2_rcu_h2_conv1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_l2_sum (Add)           (None, 24, 24, 256)  0           rb_2_rcu_l2_conv2[0][0]
                                                                 rb_2_rcu_l1_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h2_conv2 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_h2_relu2[0][0]
__________________________________________________________________________________________________
rb_2_mrf_conv_lo (Conv2D)       (None, 24, 24, 256)  590080      rb_2_rcu_l2_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_h2_sum (Add)           (None, 48, 48, 256)  0           rb_2_rcu_h2_conv2[0][0]
                                                                 rb_2_rcu_h1_sum[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 24, 24, 256)  1024        rb_2_mrf_conv_lo[0][0]
__________________________________________________________________________________________________
rb_2_mrf_conv_hi (Conv2D)       (None, 48, 48, 256)  590080      rb_2_rcu_h2_sum[0][0]
__________________________________________________________________________________________________
rb_2_mrf_up (UpSampling2D)      (None, 48, 48, 256)  0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 48, 48, 256)  1024        rb_2_mrf_conv_hi[0][0]
__________________________________________________________________________________________________
rb_2_mrf_sum (Add)              (None, 48, 48, 256)  0           rb_2_mrf_up[0][0]
                                                                 batch_normalization_16[0][0]
__________________________________________________________________________________________________
rb_2_crp_relu (ReLU)            (None, 48, 48, 256)  0           rb_2_mrf_sum[0][0]
__________________________________________________________________________________________________
rb_2_crp_conv1 (Conv2D)         (None, 48, 48, 256)  590080      rb_2_crp_relu[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 48, 48, 256)  1024        rb_2_crp_conv1[0][0]
__________________________________________________________________________________________________
rb_2_crp_pool1 (MaxPooling2D)   (None, 48, 48, 256)  0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
rb_2_crp_conv2 (Conv2D)         (None, 48, 48, 256)  590080      rb_2_crp_pool1[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 48, 48, 256)  1024        rb_2_crp_conv2[0][0]
__________________________________________________________________________________________________
rb_2_crp_pool2 (MaxPooling2D)   (None, 48, 48, 256)  0           batch_normalization_18[0][0]
__________________________________________________________________________________________________
rb_2_crp_conv3 (Conv2D)         (None, 48, 48, 256)  590080      rb_2_crp_pool2[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 48, 48, 256)  1024        rb_2_crp_conv3[0][0]
__________________________________________________________________________________________________
rb_2_crp_pool3 (MaxPooling2D)   (None, 48, 48, 256)  0           batch_normalization_19[0][0]
__________________________________________________________________________________________________
```

```
rb_2_crp_conv4 (Conv2D)         (None, 48, 48, 256)  590080      rb_2_crp_pool3[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 48, 48, 256)  1024        rb_2_crp_conv4[0][0]
__________________________________________________________________________________________________
rb_2_crp_pool4 (MaxPooling2D)   (None, 48, 48, 256)  0           batch_normalization_20[0][0]
__________________________________________________________________________________________________
rb_2_crp_sum (Add)              (None, 48, 48, 256)  0           rb_2_crp_relu[0][0]
                                                                 rb_2_crp_pool1[0][0]
                                                                 rb_2_crp_pool2[0][0]
                                                                 rb_2_crp_pool3[0][0]
                                                                 rb_2_crp_pool4[0][0]
__________________________________________________________________________________________________
rb_2_rcu_o1_relu1 (ReLU)        (None, 48, 48, 256)  0           rb_2_crp_sum[0][0]
__________________________________________________________________________________________________
rb_2_rcu_o1_conv1 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_o1_relu1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_o1_relu2 (ReLU)        (None, 48, 48, 256)  0           rb_2_rcu_o1_conv1[0][0]
__________________________________________________________________________________________________
rb_2_rcu_o1_conv2 (Conv2D)      (None, 48, 48, 256)  590080      rb_2_rcu_o1_relu2[0][0]
__________________________________________________________________________________________________
rb_2_rcu_o1_sum (Add)           (None, 48, 48, 256)  0           rb_2_rcu_o1_conv2[0][0]
                                                                 rb_2_crp_sum[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l1_relu1 (ReLU)        (None, 48, 48, 256)  0           rb_2_rcu_o1_sum[0][0]
__________________________________________________________________________________________________
resnet_map4 (Conv2D)            (None, 96, 96, 256)  65792       res2c_relu[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l1_conv1 (Conv2D)      (None, 48, 48, 256)  590080      rb_1_rcu_l1_relu1[0][0]
__________________________________________________________________________________________________
rb_1_rcu_h1_relu1 (ReLU)        (None, 96, 96, 256)  0           resnet_map4[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l1_relu2 (ReLU)        (None, 48, 48, 256)  0           rb_1_rcu_l1_conv1[0][0]
__________________________________________________________________________________________________
rb_1_rcu_h1_conv1 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_h1_relu1[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l1_conv2 (Conv2D)      (None, 48, 48, 256)  590080      rb_1_rcu_l1_relu2[0][0]
__________________________________________________________________________________________________
rb_1_rcu_h1_relu2 (ReLU)        (None, 96, 96, 256)  0           rb_1_rcu_h1_conv1[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l1_sum (Add)           (None, 48, 48, 256)  0           rb_1_rcu_l1_conv2[0][0]
                                                                 rb_2_rcu_o1_sum[0][0]
__________________________________________________________________________________________________
rb_1_rcu_h1_conv2 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_h1_relu2[0][0]
__________________________________________________________________________________________________
rb_1_rcu_l2_relu1 (ReLU)        (None, 48, 48, 256)  0           rb_1_rcu_l1_sum[0][0]
__________________________________________________________________________________________________
rb_1_rcu_h1_sum (Add)           (None, 96, 96, 256)  0           rb_1_rcu_h1_conv2[0][0]
```

```
                                                                resnet_map4[0][0]
____________________________________________________________________________________________________
rb_1_rcu_l2_conv1 (Conv2D)      (None, 48, 48, 256)  590080      rb_1_rcu_l2_relu1[0][0]
____________________________________________________________________________________________________
rb_1_rcu_h2_relu1 (ReLU)        (None, 96, 96, 256)  0           rb_1_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
rb_1_rcu_l2_relu2 (ReLU)        (None, 48, 48, 256)  0           rb_1_rcu_l2_conv1[0][0]
____________________________________________________________________________________________________
rb_1_rcu_h2_conv1 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_h2_relu1[0][0]
____________________________________________________________________________________________________
rb_1_rcu_l2_conv2 (Conv2D)      (None, 48, 48, 256)  590080      rb_1_rcu_l2_relu2[0][0]
____________________________________________________________________________________________________
rb_1_rcu_h2_relu2 (ReLU)        (None, 96, 96, 256)  0           rb_1_rcu_h2_conv1[0][0]
____________________________________________________________________________________________________
rb_1_rcu_l2_sum (Add)           (None, 48, 48, 256)  0           rb_1_rcu_l2_conv2[0][0]
                                                                 rb_1_rcu_l1_sum[0][0]
____________________________________________________________________________________________________
rb_1_rcu_h2_conv2 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_h2_relu2[0][0]
____________________________________________________________________________________________________
rb_1_mrf_conv_lo (Conv2D)       (None, 48, 48, 256)  590080      rb_1_rcu_l2_sum[0][0]
____________________________________________________________________________________________________
rb_1_rcu_h2_sum (Add)           (None, 96, 96, 256)  0           rb_1_rcu_h2_conv2[0][0]
                                                                 rb_1_rcu_h1_sum[0][0]
____________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 48, 48, 256)  1024        rb_1_mrf_conv_lo[0][0]
____________________________________________________________________________________________________
rb_1_mrf_conv_hi (Conv2D)       (None, 96, 96, 256)  590080      rb_1_rcu_h2_sum[0][0]
____________________________________________________________________________________________________
rb_1_mrf_up (UpSampling2D)      (None, 96, 96, 256)  0           batch_normalization_21[0][0]
____________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 96, 96, 256)  1024        rb_1_mrf_conv_hi[0][0]
____________________________________________________________________________________________________
rb_1_mrf_sum (Add)              (None, 96, 96, 256)  0           rb_1_mrf_up[0][0]
                                                                 batch_normalization_22[0][0]
____________________________________________________________________________________________________
rb_1_crp_relu (ReLU)            (None, 96, 96, 256)  0           rb_1_mrf_sum[0][0]
____________________________________________________________________________________________________
rb_1_crp_conv1 (Conv2D)         (None, 96, 96, 256)  590080      rb_1_crp_relu[0][0]
____________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 96, 96, 256)  1024        rb_1_crp_conv1[0][0]
____________________________________________________________________________________________________
rb_1_crp_pool1 (MaxPooling2D)   (None, 96, 96, 256)  0           batch_normalization_23[0][0]
____________________________________________________________________________________________________
rb_1_crp_conv2 (Conv2D)         (None, 96, 96, 256)  590080      rb_1_crp_pool1[0][0]
____________________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, 96, 96, 256)  1024        rb_1_crp_conv2[0][0]
____________________________________________________________________________________________________
```

```
rb_1_crp_pool2 (MaxPooling2D)   (None, 96, 96, 256)  0           batch_normalization_24[0][0]
--------------------------------------------------------------------------------------------------
rb_1_crp_conv3 (Conv2D)         (None, 96, 96, 256)  590080      rb_1_crp_pool2[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_25 (BatchNo (None, 96, 96, 256)  1024        rb_1_crp_conv3[0][0]
--------------------------------------------------------------------------------------------------
rb_1_crp_pool3 (MaxPooling2D)   (None, 96, 96, 256)  0           batch_normalization_25[0][0]
--------------------------------------------------------------------------------------------------
rb_1_crp_conv4 (Conv2D)         (None, 96, 96, 256)  590080      rb_1_crp_pool3[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_26 (BatchNo (None, 96, 96, 256)  1024        rb_1_crp_conv4[0][0]
--------------------------------------------------------------------------------------------------
rb_1_crp_pool4 (MaxPooling2D)   (None, 96, 96, 256)  0           batch_normalization_26[0][0]
--------------------------------------------------------------------------------------------------
rb_1_crp_sum (Add)              (None, 96, 96, 256)  0           rb_1_crp_relu[0][0]
                                                                 rb_1_crp_pool1[0][0]
                                                                 rb_1_crp_pool2[0][0]
                                                                 rb_1_crp_pool3[0][0]
                                                                 rb_1_crp_pool4[0][0]
--------------------------------------------------------------------------------------------------
rb_1_rcu_o1_relu1 (ReLU)        (None, 96, 96, 256)  0           rb_1_crp_sum[0][0]
--------------------------------------------------------------------------------------------------
rb_1_rcu_o1_conv1 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_o1_relu1[0][0]
--------------------------------------------------------------------------------------------------
rb_1_rcu_o1_relu2 (ReLU)        (None, 96, 96, 256)  0           rb_1_rcu_o1_conv1[0][0]
--------------------------------------------------------------------------------------------------
rb_1_rcu_o1_conv2 (Conv2D)      (None, 96, 96, 256)  590080      rb_1_rcu_o1_relu2[0][0]
--------------------------------------------------------------------------------------------------
rb_1_rcu_o1_sum (Add)           (None, 96, 96, 256)  0           rb_1_rcu_o1_conv2[0][0]
                                                                 rb_1_crp_sum[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o1_relu1 (ReLU)          (None, 96, 96, 256)  0           rb_1_rcu_o1_sum[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o1_conv1 (Conv2D)        (None, 96, 96, 256)  590080      rf_rcu_o1_relu1[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o1_relu2 (ReLU)          (None, 96, 96, 256)  0           rf_rcu_o1_conv1[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o1_conv2 (Conv2D)        (None, 96, 96, 256)  590080      rf_rcu_o1_relu2[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o1_sum (Add)             (None, 96, 96, 256)  0           rf_rcu_o1_conv2[0][0]
                                                                 rb_1_rcu_o1_sum[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o2_relu1 (ReLU)          (None, 96, 96, 256)  0           rf_rcu_o1_sum[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o2_conv1 (Conv2D)        (None, 96, 96, 256)  590080      rf_rcu_o2_relu1[0][0]
--------------------------------------------------------------------------------------------------
rf_rcu_o2_relu2 (ReLU)          (None, 96, 96, 256)  0           rf_rcu_o2_conv1[0][0]
```

```
----------------------------------------------------------------------------------------------------
rf_rcu_o2_conv2 (Conv2D)        (None, 96, 96, 256)  590080      rf_rcu_o2_relu2[0][0]

----------------------------------------------------------------------------------------------------
rf_rcu_o2_sum (Add)             (None, 96, 96, 256)  0           rf_rcu_o2_conv2[0][0]
                                                                 rf_rcu_o1_sum[0][0]

----------------------------------------------------------------------------------------------------
rf_up_o (UpSampling2D)          (None, 384, 384, 256 0           rf_rcu_o2_sum[0][0]

----------------------------------------------------------------------------------------------------
rf_pred (Conv2D)                (None, 384, 384, 3)  771         rf_up_o[0][0]
====================================================================================================
Total params: 96,750,403
Trainable params: 96,635,843
Non-trainable params: 114,560
----------------------------------------------------------------------------------------------------
```