Fine-Grained Segmentation Task for Fashion Using Fused EfficientDet and Mask R-CNN

Ofek Pearl
ofekpearl@gmail.com

Niv Amitay
nivamitay11@gmail.com

ABSTRACT

Recent advancements in the field of machine learning, specifically in tasks such as instance segmentation and detection, allow us to explore new combinations of network components in order to create hybrid networks that achieve better results. In this work, we propose a novel network composed of three components responsible for solving classification, detection, and instance segmentation. Since achieving state-of-the-art performance is unlikely with limited resources, we aim towards increasing the efficiency of our model compared to the baseline model without sacrificing accuracy. Our approach is based on Mask R-CNN [7] architecture for segmentation, EfficientNet [24] as a backbone, and EfficientDet [25] as a bounding box and classification branch which also serves as an RPN for the segmentation branch. As far as we know this is the first time this combination has been attempted. Our model achieves comparable accuracy results on the Fashionpedia dataset [9] compared to the baseline network which is a standard Mask R-CNN network, while being up to 3.9x smaller (see table 1) and is up to 7.34x faster on inference using CPU (see table 2). By efficiently scaling up our model, we were also able to improve the bounding box accuracy by 20% and the segmentation accuracy by 8% compared to the baseline network while still obtaining a network with 2.3x less parameter (see EfficientDet d2 in table 1).

Code:
The source code of this paper has been made publicly available at https://github.com/ofekp/imat. To avoid duplicating code from standard libraries and allow for a more maintainable codebase we've forked two other repositories and made changes to them while compiling them manually. For Efficient-Det we used [27] and our changes to this library are available in this link https://github.com/ofekp/efficientdet-pytorch code comparison is available here https://github.com/rwightman/efficientdet-pytorch/compare/master...ofekp:master. For Mask R-CNN we used [18] and our changes to this library are available in this link https://github.com/ofekp/vision code comparison is available here https://github.com/pytorch/vision/compare/master...ofekp:master.

1 INTRODUCTION

Designers and fashion artists who create the clothes we wear understand their designs very well but lack the information on how their designs are being used and how they are combined with other items. In recent years, the experience of buying fashion apparel has changed dramatically, transitioning from buying at physical stores to buying online. In 2019 in the US more than 38.6% of apparel purchases were made online with the numbers expected to rise. According to Statista [16], the global online fashion market was worth 533$ billion in 2018 and is predicted to grow to 872$ billion by 2023. Customer behavior also indicates that online research is an important part of the purchase even if the purchase was made at a physical store. It is no wonder that fashion retailers and shops are interested in machine learning algorithms that will be able to predict trends and will be more capable of recommending items to customers. Fashion artists are also interested in the data that machine learning algorithms are capable to produce to find out how their products are being used, as well as analyze customer usage.

The increasing amount of fashion data available online which includes both fashion items and fashion models enables algorithms to be more accurate at detecting and classifying clothing items. Accurate product recognition is vital to achieve good signal for recommendation models and accurate data to fashion professionals that want to design better and more up-to-date apparel items.

As part of the Fine-Grained Visual Categorization (FGVC7) workshop at the Conference on Computer Vision and Pattern Recognition (CVPR) 2020, the iMaterialist [26] competition was established on Kaggle by Google AI to inspire further advancement by challenging competitors to devise new algorithms that will be more accurate at recognizing clothing items. For each apparel, the competitors must mark all apparel parts that were recognized by the model. For example, a shirt may have two sleeves, a collar, and a pocket, all of which, including the shirt itself, are identified, bounded and segmented separately. In addition, two instances of the same class should be recognized as different objects e.g., two instances of shoes worn by the same person are considered two different objects.

Advancements in both detection and segmentation models have increased the potential of using machine learning techniques to the benefit of the fashion industry. Much of these advancements are due to the rise of deep neural networks. We propose a novel network to achieve both segmentation and detection of clothing items. We apply a recent instance segmentation technique [7] and recent detection technique [25] to solve three seemingly separated problems: 1) identify the classes of each instance of clothing in a picture; 2) apply a bounding box that will mark its margins; 3) recognize its pixel segmentation with fine accuracy. Our ultimate goal is to provide a deep neural network (DNN) that will tackle the iMaterialist challenge with comparable results to a state-of-the-art model while achieving better efficiency through a DNN with fewer parameters that will be more manageable by edge-devices such as mobile phones/smart mirrors.

In this work, we show our model has reached comparable results to those of the baseline model for both bounding box and segmentation predictions while being up to 3.9x smaller. Furthermore, we show that increasing the model size using the paradigm described by both EfficientNet [24] and EfficientDet [25] papers will allow for improvement in the results of both the detection and segmentation compared to the baseline model. Specifically, using EfficientNet b2 and EfficientDet d2 we reach a significant improvement and still get a model which is 2x smaller (Table 1) and 4.83x faster on CPU (Table 2.

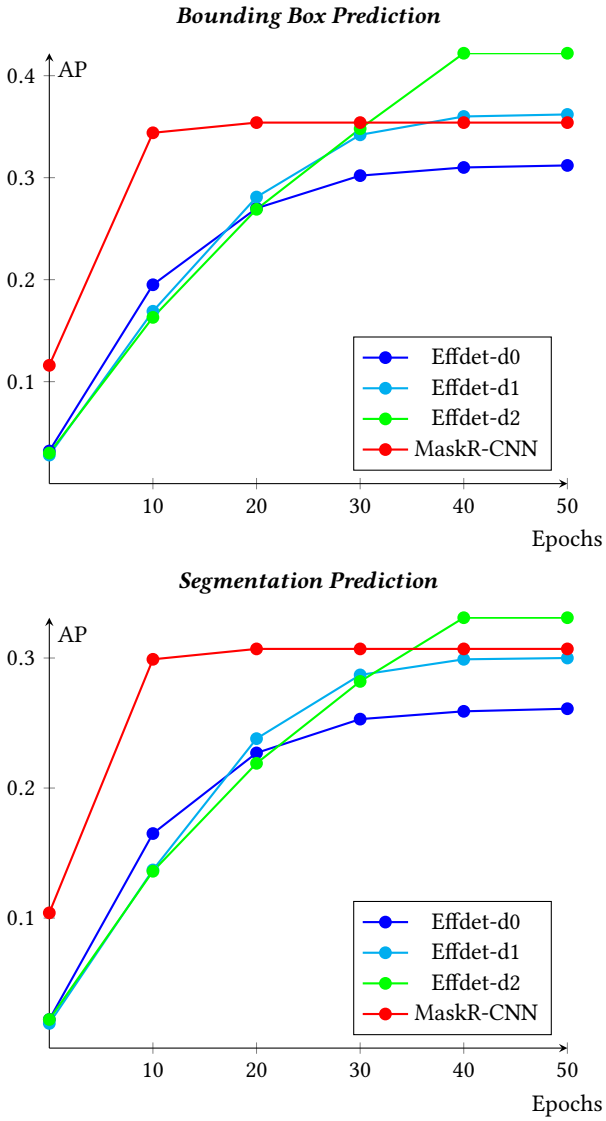

Figure 1: Comparing our model variants, called Effdet-d0/d1/d2 in the figure, to a baseline Mask R-CNN model. We show that while our model variants are more efficient and contain significantly fewer parameters, we reach comparable results. Furthermore, we show that by scaling up our network, we produce better results.

2 DATASET

Our dataset relies on the Fashionpedia [9] dataset. The main purpose of the Fashionpedia ontology and fashion segmentation dataset is to map visual fashion aspects from real-world images with fine categorization consisting of 46 apparel objects and their relationship with the aim of training and benchmarking computer vision models for a more comprehensive understanding of fashion. The Fashionpedia ontology relies on the notions of objects that represent items in apparels and statements that describe the characteristics of an object. The dataset further defines three types of relationships: 1) part-of relationship - i.e. outfit to main garment or main garment to garment parts; 2) attributes relationship – both main garments and garment parts can be assigned with attributes; 3) instance-of relationship – with a max of 4 levels which apply to both garment parts and attributes. In our dataset, every image is annotated with main garments and garment parts, and every garment and garment part is annotated with attributes. Main garments are divided into three main categories: outerwear, intimate, and accessories. For example, "pants" are mapped as a main garment part and can be linked with "button" or "pocket" as the apparel part and each of them can have different material attributes. The data is divided with the fine-grained attributes method, meaning each main garment and garment part were associated with apparel attributes, for example, “button” is the part of the main garment “jacket” can be linked with the silhouette attribute “symmetrical”, garment part “Button” could contain attribute “metal” with a relationship of material. Each image in Fashionpedia dataset has on average, 1 person, 3 main garments, 3 accessories, 12 garment parts, and 16 attributes. From the objects and the statements, we can create an outfit ensemble that represents the structure of the different segments in the image: main garments and garment parts are linked to their respective attributes through different types of relationships (refer to graph in figure 2). The main purpose of creating ensembles of outfits and representing the data in a graph is to create and combine multiple levels of information for every image in a more coherent way.

The dataset contains 48,827 images collected from fashion websites and fashion retailers’ websites and was tagged in 3 steps: 1) two fashion experts verify the quality of the images manually; 2) the annotation process starts with segmentation masks of apparel objects by crowd workers using the image description on the website; 3) 15 fashion experts annotated the fine-grained attributes for the segmentation masks. The segmentation masks may be contained in each other, for example, if there is a pocket on a shirt, the pixels of the pocket will be shown in both the shirt and the pocket segmentation masks. Such overlap in segmentation masks is only possible for objects that have a part-of relationship.

We concentrate our work mainly on the segmentation and classification of the main garments and garment parts with the main 46 apparel classes. The masks in the dataset are encoded in a compressed way which had to be decoded into an image before they could be used as an input to the models or presented for visualization. To visualize the dataset and the predictions of the models during inference, we implemented a method to show the bounding boxes and segments. The method gets an image, a list of bounding boxes, and a list of segmentation masks for each object and plots it on the original image. Our implementation relies on the Bounding Box library [15] to draw the bounding boxes. We used our own implementation for drawing the segmentation masks. Our implementation allows us to plot the different mask segments together in one image or separately for each mask segment. In both cases, the segments are plotted on the original image and assigned transparency for convenient observation of the data annotations and model predictions, see figure 3 for an example.

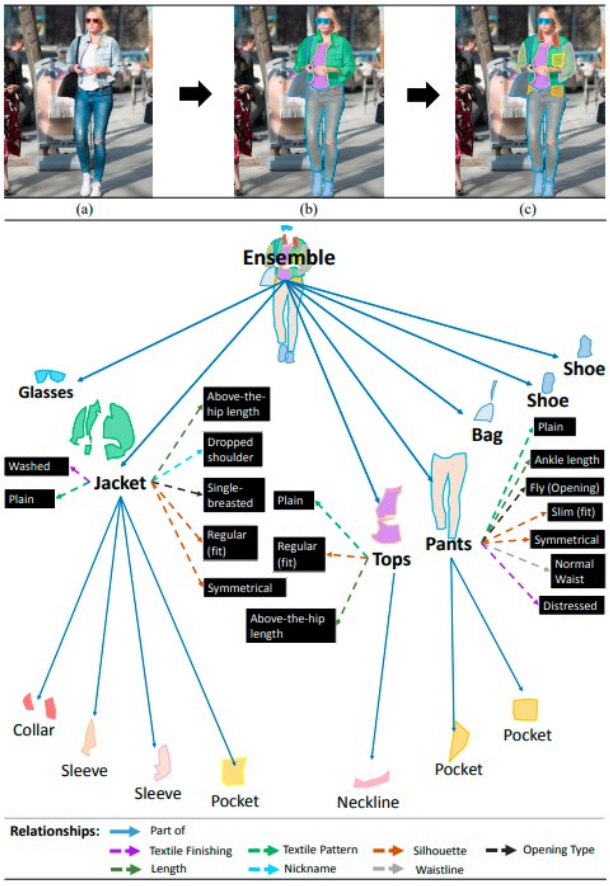

Figure 2: Overview of The dataset, at the top we can see the original image (a), the first step of main garment segmentation (b), and the final step with both the main garment and garment segmentation (c). At the bottom, we can see the annotation diagram including the mask, attributes, and corresponding relationships.

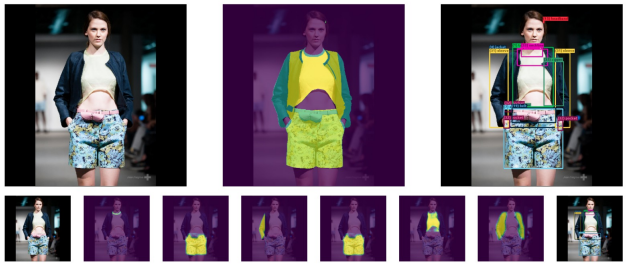

Figure 3: Example of plotting the data using our implementation: the top left image is the original image, the top center image is the image with all the segments drawn on it, and on the top right is the image with all the bounding box annotations. The bottom image contains our predictions of mask segments but displays the mask segments separately.

3 RELATED WORK

Instance segmentation is one of the relatively important, complex, and challenging areas in machine learning based computer vision algorithms and is of great importance in many fields of our life, from the automotive industry to medical applications. In segmentation problems, we are trying to predict the object class label and the pixel specific mask. Instance segmentation is a step up from segmentation in the sense that objects of the same class should be recognized as different instances. In this section, we will introduce state-of-the-art work for instance segmentation and detection and then we will focus on the iMaterialist competition winners in the last 2 years, their approach, and methods to tackle the iMaterialist challenge.

3.1 Mask R-CNN

Two-stage segmentation networks consist of an object detector prior to the segmentation stage. To name a few of the recent advancements in two-stage based detection networks: R-CNN [6]; Fast R-CNN [5]; Faster R-CNN [21]. Other detection networks use a single-stage architecture e.g., YOLO [20]. The main disadvantage of the two-stage method is that it is more computationally demanding in both training time and inference, on the other hand, it usually leads to more accurate detection. Mask R-CNN [7] is one of the leading networks in instance segmentation. It falls under the category of two-stage detectors where the first stage is Region Proposal Network (RPN) first suggested in [21] paper, which makes the second step of the segmentation part more manageable since it works on the proposed regions. Mask R-CNN uses Faster R-CNN as its RPN and produces a binary mask for each segment in that second step. At its core, Mask R-CNN also makes use of a backbone combined with Feature Pyramid Network (FPN) [10] to detect the features of the image. The FPN uses a top-down approach for gathering the semantically strong features on the different scales of the image.

3.2 iMaterialist 2019 Winner

The winner of 2019 is Miras Amir [1]. Miras' model was based on the paper Hybrid Task Cascade for Instance Segmentation [2]. For the preprocessing stage and the data preparation, the author emphasizes the importance of applying different augmentations for avoiding overfitting. The paper Hybrid Task Cascade for Instance Segmentation describes the main difference from a regular instance segmentation task solution such as Mask R-CNN. The differences are mostly in the fact that the bounding box stage and the segmentation stage are done consecutively rather than in parallel and the flow of information allows for direct connections between the stages. Using the Hybrid Task Cascade, the author is applying Test Time Augmentation where each image is randomly uniformly scaled multiple times. Each scaled image augmentation is given to the same model and non-maximum suppression (NMS) is used after the RPN and bounding box steps to reconcile the results. For the segmentation, mean is applied to reconcile the results of the last step of the network. The author mentioned that he did not predict the attributes since the final score did not take those into consideration.

3.3 iMaterialist 2020 Winner

The winner of 2020 is Oleg Polosin [17]. Oleg's model was based on a single Mask R-CNN with a Spine-Net-143 [3] combined with an FPN [10] as a backbone, and an extra head for attribute detection. For augmentation, Oleg used the V3 policy from Google team AutoAugment policies, depicted in the paper Learning Data Augmentation Strategies for Object Detection [28]. The author used the backbone and FPN architecture to extract the ROIs which were then used by three other head branches: 1) bounding box and class predictor exactly as it is implemented in the Mask R-CNN model; 2) segmentation predictor from the baseline Mask R-CNN model; 3) the author added a fully connected network to predict the attribute of the object; For the 2020 competition, the evaluation metric was changed. Compared to 2019 competition attributes were now added to consideration in the final F1 score. The winner used the complete dataset with all images resized to 1280x1280 resolution for the training process while allocating 5,691 images as the test set. The author trained the model for 3 days using TPUs on the Google Cloud Platform (GCP).

4 METHODS

Object detection and instance segmentation tasks have been improved lately with the appearance of Mask R-CNN [7]. In our work we decided to take advantage of yet more recent advancements in both detection [25] and feature extraction [24] to improve on the work of Mask R-CNN. Mask R-CNN architecture consists of a feature extraction backbone, an FPN, and 2 branches, one for bounding box and classification and another for segmentation. We present a novel model that achieves comparable and even improved results while being smaller, achieves faster inference times, and is faster to train. Based on the Mask R-CNN architecture we replace the traditional backbone of ResNet [8] with EfficientNet [24] which gained a lot of attention by achieving state-of-the-art results on the COCO dataset [11]. Furthermore, we replace the FPN [4] which feeds both the detection and segmentation branches with the innovative bi-directional feature pyramid network (BiFPN) proposed in EfficientDet paper [25]. Lastly, we replace the detection and classification branch [21] with the recently proposed leading detection network EfficientDet. We kept the original segmentation head as is. Figure 4 shows the complete architecture of our model after those modifications.

4.1 Backbone

The first and one of the most important tasks of detection and segmentation is the extraction of relevant features from the image on different scales. For that purpose, we chose to rely on the same backbone used by EfficientDet [25], which is EfficinetNet [24] backbone. EfficientNet paper lays out a family of ConvNets that can be scaled easily and uniformly in all the dimensions of depth/width/resolution using a simple yet highly effective compound coefficient. Using MnasNet [23] and a new definition of the optimization goal as a combination of the accuracy and FLOPs they reached a baseline network called EfficientNet-B0. From that baseline, the authors were focusing on finding a principled method to scale up networks that can achieve better accuracy and efficiency and proposed the compound scaling method. Basing their suggestions on two main observation: 1) scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models; 2) in order to pursue better accuracy and efficiency, it is critical to balance all the dimensions of network, width, depth, and resolution during ConvNet scaling. They proposed the next compound scaling method to scale the network from the baseline:

$$\begin{aligned} depth &: d = \alpha^{\phi} \\ width &: w = \beta^{\phi} \\ resolution &: r = \gamma^{\phi} \\ s.t &: \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ \alpha &\geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{1}$$

α, β, γ are constants that can be determined by a small grid search and ϕ is the compound coefficient where ϕ=1 corresponds to b1 up to ϕ=7 which corresponds to b7 architecture.

4.2 BoundingBox and Classification Branch

For the bounding box predictor and classification predictor, we chose to work with EfficientDet [25] network. EfficientDet network was developed by the Google Research team based on the EfficientNet [24] with the goal to achieve much better efficiency from the object detector while not sacrificing its accuracy. Inspired by EfficientNet, the authors devised a scaling method that scales uniformly the width, depth, and resolution for all network components – backbone, BiFPN, bounding box, and class prediction networks with a single compound coefficient. EfficinetDet also aims to provide a more efficient multi-scale feature extraction and fusion component by replacing the traditional, single direction FPN [10] with the BiFPN which introduces learnable weights to learn the importance of different input features at different scales. The BiFPN is seeking a transformation f that can effectively assemble different features from different resolutions and levels as received from the backbone, $\vec{P^{out}} = f(\vec{P^{in}})$ where $\vec{P^{in}} = (P^{in}_{l1}, P^{in}_{l2}, \dots)$ and $P^{in}_{l_i}$ is a feature at level l_i. A previous related works for feature extraction like FPN [10], PANet [12] and NAS-FPN [4] tried to address the feature extraction in various ways. In the traditional FPN [10] used by Mask R-CNN [21] the method takes one single-scale image and outputs proportionally sized feature maps at multiple levels. The authors used a top-down process that takes the image and creates a resolution pyramid where P^{in}_i represents a feature level with a resolution of $1/2^i$ of the input image, as an example, for input image with a resolution of 512x512, P^{in}_4 represents feature level 4: $(512/2^4 = 32)$ and the last $7th$ level yields a 4x4 resolution. The multi-scale features assembly is made by this top-down manner in a recursive method in the following manner: $P^{out}_7 = Conv(P^{in}_7); P^{out}_6 = Conv(P^{in}_6 + Resize(P^{out}_7))$ and so on, down to the base level, where Resize is upsampling the spatial resolution by a factor of 2 using nearest-neighbor interpolation. This FPN method is very limited due to the one-way flow of information, so to address this problem, PANet [12] adds another bottom-up layer for the aggregation in the opposite direction. Another method is to apply a search for cross-scale connections for finding better cross-scale feature network topology as proposed in NAS-FPN [4],

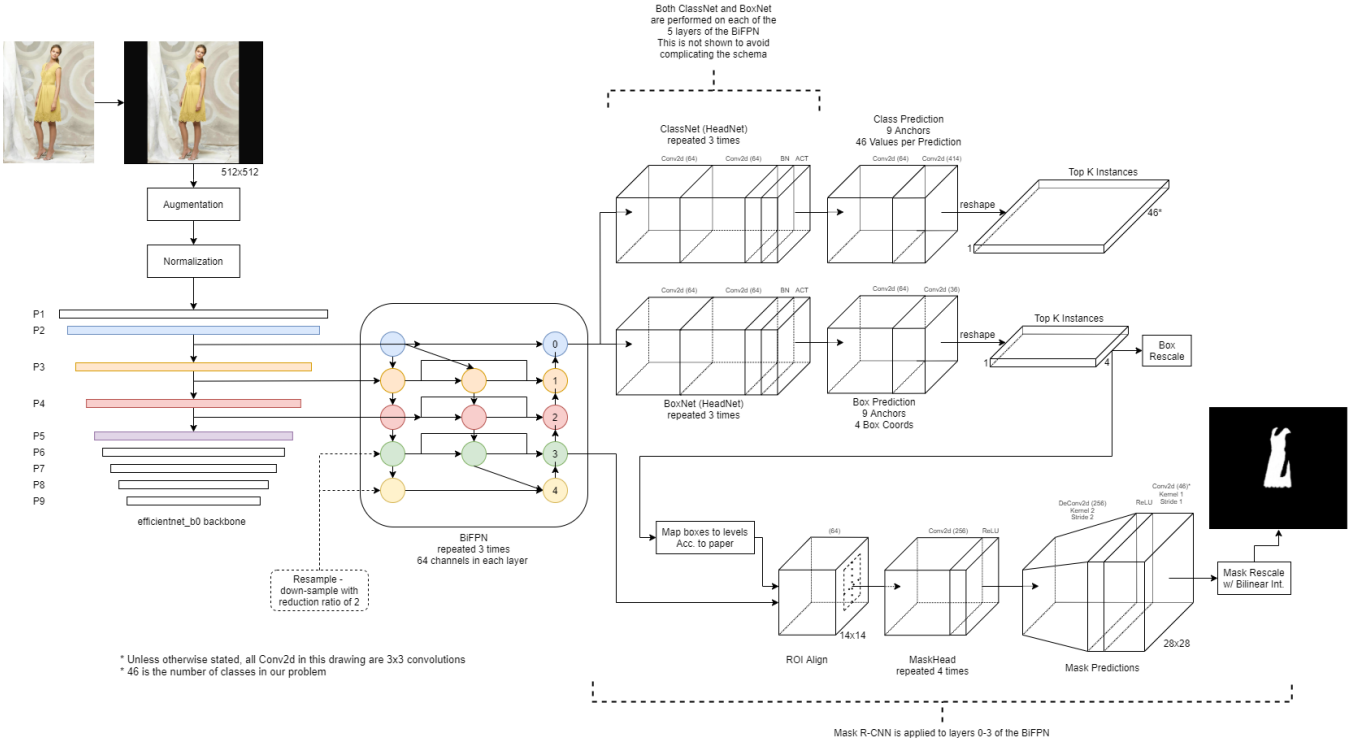

Figure 4: Our modified Mask R-CNN based network architecture. We replaced the backbone from ResNet to EfficientNet. The FPN is replaced with BiFPN that was proposed in the EfficientDet paper. We also replace the entire detection branch from Faster R-CNN to EfficientDet which now uses the same backbone and BiFPN as the segmentation branch. The BiFPN completes the 5 levels shown in the figure by down-sampling the last feature layer (4) two more times, in each level, the size of the feature map is decreased by a factor of 2. Then, this BiFPN layer is repeated 3 times for the d0 model variant. The detection branch input is all the outputs of the BiFPN while the segmentation branch is only processing layers 0-3 of the BiFPN output. In the detection branch, we have the BoxNet and the ClassNet, both are similar up to final convolution where the number of channels differs, after that a reshape and top k extraction will give us the boxes and classes. The boxes are then fed to the segmentation network after they are divided into the respective levels based on spatial rules, from here we have a standard Mask R-CNN network with ROI Align, mask head and then mask prediction to give the final mask for each box prediction.

but this method requires great computational power and a long search time. BiFPN proposes several optimizations for cross-scale connections: 1) removing middle nodes that have only one input edge (Figure 4); 2) adding an edge from the original input to output node if they are at the same level with the intent to fuse more features without adding too much cost of resizing transformations; 3) treating each bidirectional path as one feature network layer and repeat the same layer multiple times to enable a more high-level feature fusion. Furthermore, BiFPN introduces a weighted sum of the features and lets the network learn the importance of each input feature. Based on the weighted sum method, BiFPN uses the fast normalized fusion: $O = \sum_i \frac{\omega_i}{\epsilon + \sum_j \omega_j} \cdot I_i$, where ω_i is a learnable weight, I_i is the input feature at level i, and ϵ is a small value to avoid numerical instability (0.0001). This method has a very similar learning behavior and accuracy as the other fusion methods (e.g. softmax) but runs up to 30% faster. The final BiFPN has integrated the bidirectional cross-scale connections and the fast normalized fusion. The next example shows the fusion of two features at level 6:

$$
\begin{aligned}
P_6^{td} &= Conv\left(\frac{\omega_1 \cdot P_6^{in} + \omega_2 \cdot Resize(P_7^{in})}{\omega_1 + \omega_2 + \epsilon}\right) \\
P_6^{out} &= Conv\left(\frac{\omega_1' \cdot P_6^{in} + \omega_2' \cdot P_6^{td} + \omega_3' \cdot Resize(P_5^{out})}{\omega_1' + \omega_2' + \omega_3' + \epsilon}\right)
\end{aligned}
\tag{2}
$$

P_6^{td} is the intermediate feature at level 6 on the top-down pathway, and P_6^{out} is the output feature at level 6 on the bottom-up pathway and also serves as the output to the next BiFPN layer or the rest of the network. Similarly, we can construct all the other features.

EfficientDet is based on a one-stage paradigm and is using EfficientNet backbone pre-trained on ImageNet [22] and BiFPN which takes the features from levels 3-7 of the backbone and applies the bidirectional feature fusion. Aiming at solving the segmentation problem, and as proposed by the EfficientDet authors in future work for segmentation, we decided to use features from higher resolution levels - 2, 3, 4, and then simple downsampling was used (Figure 4).

At the last step, the BiFPN output is fed into a class network and bounding box network. As previously stated EfficientDet authors devised a scaling paradigm to adjust the width, depth, and resolution of the network. The method relies on a compound coefficient ϕ to jointly scale-up all dimensions in the different modules as detailed: 1) the EfficinetNet backbone is scaled according to the coefficients as shown in equation 1 in section 4.1; 2) the BiFPN depth is increased linearly and its width is changed by picking the best value from a grid search. In total we get the width and depth by: $W_{BiFPN} = 64 \cdot 1.35\phi; D_{BiFPN} = 3 + \phi$; 3) in the class and bounding box networks, the width is always the same as in the BiFPN but the depth is changed by the following equation: $D_{box} = D_{class} = 3 + \lfloor \frac{\phi}{3} \rfloor$; The input resolution is determined by the next equation and ensures a resolution that can be divided by $2^7 = 128$ to reach the $7th$ level in the BiFPN:

$$Res_{input} = 512 + \phi \cdot 128 \tag{3}$$

ϕ=0 for EfficientDet d0, up to ϕ=7 for EfficientDet d7. EfficientDet achieves the same accuracy as other widely-used detection models, but is 4x - 9x smaller, 2x - 4x faster on GPU, and 5x - 11x faster on CPU compared to other detectors.

4.3 Segmentation Branch

For the segmentation head, we used part of the Mask R-CNN [7] architecture. Mask R-CNN adopts Faster R-CNN two-stage procedure, with an identical first stage - the Region Proposal Network (RPN) proposes Regions of Interest (ROI), and the second stage consumes those ROIs to produce bounding box and class predictions. On top of the detection branch, Mask R-CNN adds a binary mask predictor, which is the part we use in our model. In our implementation, the segmentation branch gets the proposed ROIs from the detection branch 4.2 and then we apply the ROI Align method proposed by Mask R-CNN authors to avoid harsh quantization and properly aligning the extracted features. The main difference from the ROI Pool which is used in Faster R-CNN for detection is that ROI Align is avoiding quantization of the ROI boundaries or bins which does not work well for segmentation. Using a per-pixel sigmoid and define mask-loss as the average binary cross-entropy loss, the network can generate masks for every class without competition among classes. The mask predictor predicts an m × m mask from each ROI using an FCN [13], which allows each layer in the mask branch to maintain the explicit m × m object spatial layout without collapsing it into a vector representation that lacks spatial dimensions, requires fewer parameters, and is more accurate than previous works.

5 EXPERIMENTS

We trained 4 models in total, our architecture is based on EfficientDet d0, d1, and d2 variants and a baseline model which is the non-modified Mask R-CNN network based on Faster R-CNN. We will refer to the former simply as “Effdet d0/d1/d2” or “Effdet” as a whole and the latter will be referred to as the “baseline model”.

5.1 Data

To train all 4 models we used 12.5k images split into 10K for train-set and 2.5K for the test-set which together make up for 25% of the Fashionpedia dataset. All 46 main garment classes from the Fashionpedia dataset were used in our training. We transformed all images to a fixed, square size of 512x512. For each rescaled image we also rescaled its annotations – segmentation masks were decoded and rescaled to the same size using nearest-neighbor interpolation, which preserves the binary nature of these mask images and the bounding boxes were also scaled accordingly for every segment in the image. For augmentations, we used a 10% probability grey-scale filter and a 50% probability horizontal flip transform.

During the training process we recognized that a lot of time was invested in each epoch on the following tasks: 1) resizing of the image; 2) decoding of each mask into an image from its compressed form in the Fashionpedia dataset; 3) resizing each decoded mask; 4) resizing the bounding boxes. In our tests, this could amount to 1.5 seconds per image which could take hours during the training. To address this issue we decided to prepare the data in advance using a package called H5PY [14] which uses Hierarchical Data Format 5 (HDF5) to save very large files with minimal access time since it allows direct access to parts of the file without first parsing the entire contents. We avoided compression of the data since that increased the processing time per image to an unusable amount. In total, 10k train images mapped to 242GB of H5PY file, and 2.5K test images were mapped to a 61GB H5PY file. This reduced training time significantly by reducing the average process time per image to 0.1 seconds. To create the H5PY files we used 5 processors and a single writer single queue methodology and were able to create the train set H5PY file in about 3 hours. To further increase efficiency we used a local SSD on our Google Cloud Platform (GCP) instance.

In an attempt to train on all the dataset we approached Tensorflow Team and were granted limited free TPU access. Since our work was done using PyTorch we encountered compatibility issues with TPU devices. Specifically, NMS and ROI Align were not supported with TPU devices in torchvision [18] package. While we were able to replace the NMS implementation in torchvision with a TPU supporting implementation from Torch XLA package [19], we were unable to find an implementation for ROI Align that supported TPU devices. We eventually recognized that this effort consumes too much time to overcome.

5.2 Training

Using AdamW optimizer we trained for 50 epochs with a learning rate of 0.01, weight decay of 5e-4, and a batch size of 12 images. We trained our Effdet-d0 variant using torch’s StepLR scheduler with a step size of 10 and a factor of 0.2. All other models were trained using torch’s ReduceLROnPlateau scheduler with a factor of 0.5 and min LR of 1e-8. Training all our Effdet variants took approximately 24 hours per variant and training the baseline Mask R-CNN network took approximately 2 days. To address the issue of the relatively low amount of GPU memory, in our case Tesla T4 with 15GB of memory, we use gradient accumulation of 2 which effectively increases the batch size to 24 and we also freeze the batch normalization weights since we gain more from the pre-trained batch normalization weights which were trained on much larger batch size. Based on the graph in Figure 1 we can see that the models start to converge when reaching 50 epochs and we noticed that even when we train for 140 epochs in some attempts, we do not reach higher results.

Training on Tesla T4 we encountered training issues such as running out of memory with batch sizes that are too big (e.g. batch size of 16 was already too big to fit into memory), in addition, to avoid training instabilities we added gradient clipping to our training process.

5.3 Results

For evaluation we use mean average precision (mAP) at the different intersection over union (IoU) thresholds which takes into account the class predictions similarly to COCO API [11] for both the masks and the bounding boxes predictions.

Table 1 shows our results of Effdet variants and the baseline model including the final model size for comparison. In our tests, the modification to Mask R-CNN yielded a smaller model without compromising the results. It is evident that the d0 model achieved close results while being 3.9x smaller than the baseline Mask R-CNN model. Moreover, when using d2 we were able to achieve 2.4% improvement in segmentation and 6.8% improvement in bounding boxes predictions with a model that is still 2.3x smaller compared to the baseline model. We did not have the capacity to check higher ϕ values which correspond to EfficientDet d3 or higher. We also use the same pre-processed H5PY dataset 5.1 to train all models, but it is worth mentioning that starting from Effdet d1 the expected resolution of the images is increasing from 512 to 640 and above, yet our base image resolution remains 512x512. Given that, we could have gained more from the larger networks by also increasing the resolution of our images.

We also wanted to test and compare the inference performance of our model to that of the baseline Mask R-CNN model. We tested the average time for evaluation in both GPU and CPU and found that our model is up to 1.35x faster on GPU and 7.34x faster on CPU when comparing to the baseline model achieving a max of 10.85 fps on GPU and 3.29 fps on CPU. We demonstrate these results in Table 2.

AP and Model Sizes Comparison				
	EfficientDet d0	EfficientDet d1	EfficientDet d2	Mask R-CNN
Box	31.2	36.2	42.2	35.4
Seg	26.1	30.0	33.1	30.7
Model Size	87M	124M	146M	337M

Table 1: Comparing AP results and model size of our model to the baseline model across multiple model sizes, it is evident that we achieve comparable results to the baseline Mask R-CNN model while our models were 2.3x to 3.9x smaller

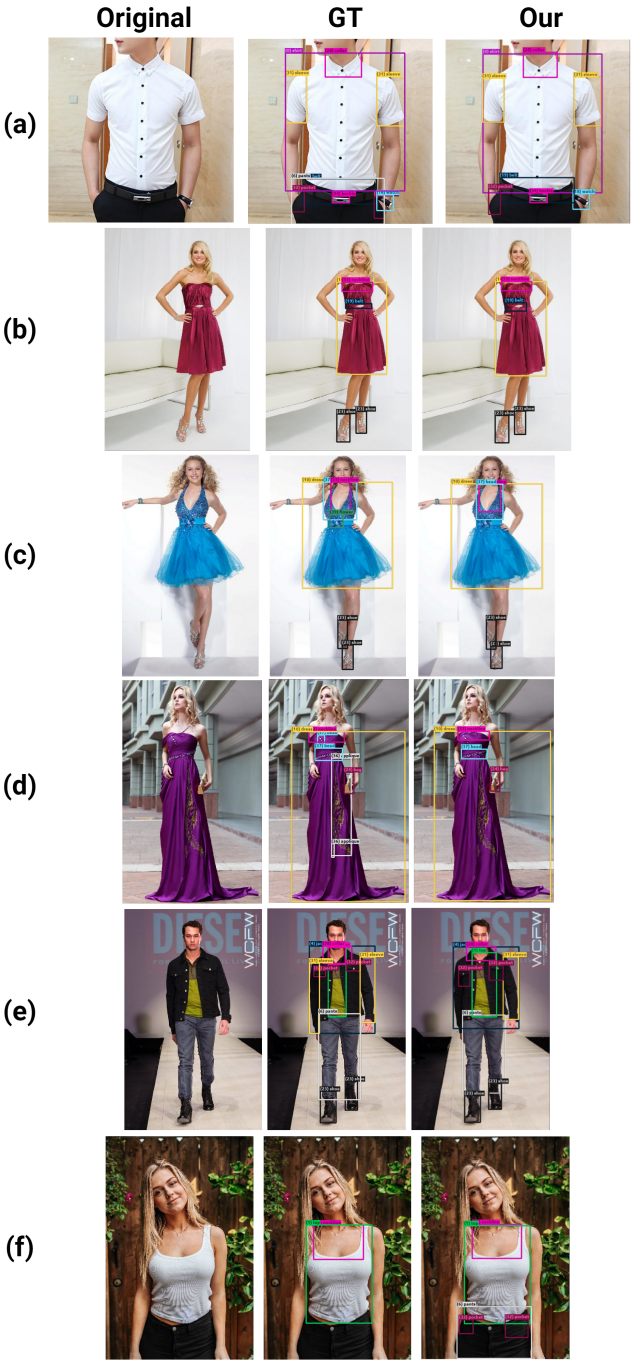

Figure 5: Selected examples showing bounding boxes predictions compared to the ground truth (GT). We discuss the results at the end of the Experiments section 5

5.4 Qualitative Analysis

In Figure 5 we selected a few examples showing bounding boxes predictions compared to the ground truth (GT). You can see exact matches in examples (e) and (b), moreover, (e) is an example of an image that contains many items (9 items in total) worn by the same person and found in close proximity to each other. In example (c) a flower on the dress was missed by our network, probably due to the low number of examples in our dataset. In example (f) we can see that even though the pockets and the pants were not marked by the GT, our network still managed to detect them. In (a) we show

Evaluation Time Comparisons						
	GPU			CPU		
Model	Eval time [sec]	rate	ratio	Eval time[sec]	rate	ratio
Mask R-CNN	0.125	8 fps	1.0x	2.234	0.44 fps	1.0x
EfficientDet d0	0.092	10.85 fps	1.35x	0.304	3.29 fps	7.34x
EfficientDet d1	0.109	9.17 fps	1.14x	0.414	2.41 fps	5.39x
EfficientDet d2	0.116	8.62 fps	1.07x	0.462	2.16 fps	4.83x

Table 2: Comparing the evaluation time of the baseline model to our EfficientDet based model variants. We used 2K images when testing on GPU and 100 images when testing on CPU. We show that our models are 1.07x to 1.35x faster on GPU, and 4.83x to 7.34x faster on CPU. Expectedly, It is evident that the difference is more significant for CPU based devices.

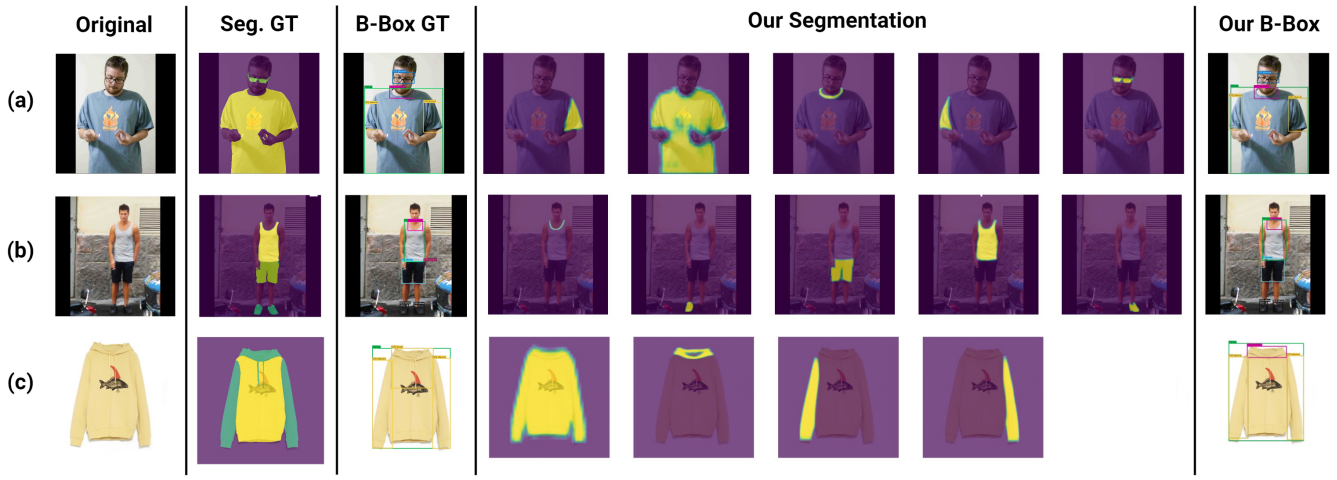

Figure 6: Selected examples to demonstrate fine grain segmentation performed by our network. We present the results while splitting the segments to separate mask images to show that the network differentiates between objects. Our network produced fine-grained masks that are on par with the GT masks, detecting and categorizing sleeves, shirts, shoes, necklines, sunglasses, and more by coloring the correct pixels.

that our network is capable to detect small items such as a watch and a buckle.

In Figure 6 we selected a few examples to demonstrate the segmentation predictions that our network produced. We show that our network can detect small items such as sunglasses and in other examples, the network distinguishes between watches or pockets and categorize their pixels in the image in detail.

6 FUTURE WORK

We would like to investigate several possible improvements as future work. The first is to train on the COCO dataset [11] and perform a decent comparison to state-of-the-art models. Other improvements that can be made are using the entire dataset, adjusting the image resolution to fit the intended resolution according to the EfficientNet scale variant, and using more augmentations such as random salt, random resizes, and random cropping, or use AutoAugment policies. During our training we also saw that increasing the batch size is beneficial to the network, probably due to the batch normalization that is done by EfficientDet but due to the limited resources we could not fit a large batch size into the GPU memory, a larger batch size may further improve the results. At this point, we could not test the effect of each component of our modified architecture on the performance of the network. To that end, separate models would have to be made in order to separately assess the performance improvement gained by each modification. Specifically, we would have liked to separately check BiFPN, EfficientNet, and EfficientDet. We would also like to investigate our model performance compared to a network that is completely relying on EfficientDet for segmentation which was proposed by EfficientDet paper authors. Finally, we would love to attend the next competition and compare our network to that of other competitors on the Kaggle platform.

7 CONCLUSION

In this paper, we propose a solution to the iMaterialist competition based on the Fashionpedia dataset. Our solution extensively modifies Mask R-CNN to use the recent advancements of EfficientNet, BiFPN, and EfficientDet and achieves comparable accuracy to that

of the baseline Mask R-CNN model while being 2.3x to 3.9x smaller, 4.83x to 7.34x faster on CPU, and 1.07x to 1.35x faster on GPU. When comparing to previous competitions [3.3, 3.2] our Effdet d2 achieves results that would place our network within the top 5 competitors using a quarter of the data. We also recognize that there is room for training networks that are based on EfficientDet d3 or higher. As is, our solution fits well for edge devices such as mobile phones and smart mirrors which may run such algorithms and have limited resources available with the hope of improving the shopping experience for users. In the future, such models may lead to extracting better features for recommendation models in major retail and online shops. In addition, product and attributes recognition can lead to improving fashion professionals' work processes and improving customer experience.

REFERENCES

[1] Miras Amir. 2019. *The First Place Solution of iMaterialist (Fashion) 2019.* https://github.com/amirassov/kaggle-imaterialist

[2] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. 2019. Hybrid task cascade for instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition.*

[3] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. 2019. SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization. arXiv:1912.05027 [cs.CV]

[4] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. 2019. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. arXiv:1904.07392 [cs.CV]

[5] Ross Girshick. 2015. Fast R-CNN. arXiv:1504.08083 [cs.CV]

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 [cs.CV]

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. arXiv:1703.06870 [cs.CV]

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[9] Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Claire Cardie, Bharath Hariharan, Hartwig Adam, and Serge Belongie. 2020. Fashionpedia: Ontology, Segmentation, and an Attribute Localization Dataset. In *European Conference on Computer Vision (ECCV).*

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2016. Feature Pyramid Networks for Object Detection. arXiv:1612.03144 [cs.CV]

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. Microsoft COCO: Common Objects in Context. arXiv:1405.0312 [cs.CV]

[12] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. 2018. Path Aggregation Network for Instance Segmentation. arXiv:1803.01534 [cs.CV]

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2014. Fully Convolutional Networks for Semantic Segmentation. arXiv:1411.4038 [cs.CV]

[14] Elena Pourmal Gerd Heber Mike Folk, Quincey Koziol. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the 2011 EDBT/ICDT Workshop on Array Databases, Uppsala, Sweden, March 25, 2011.* https://doi.org/10.1145/1966895.1966900

[15] Manu NALEPA. 2019. *Bounding Box Library.* https://github.com/nalepae/bounding-box

[16] Liam O'Connell. 2020. *Global Apparel Market - Statistics Facts.* Statista. Retrieved January 01, 2020 from https://www.statista.com/topics/5091/apparel-market-worldwide/

[17] Oleg Polosin. 2020. *The First Place Solution of iMaterialist (Fashion) 2020.* https://www.kaggle.com/c/imaterialist-fashion-2020-fgvc7/discussion/154306

[18] PyTorch. 2017. *PyTorch Vision Library.* PyTorch. Retrieved September 20, 2020 from https://github.com/pytorch/vision

[19] PyTorch. 2020. *PyTorch/XLA.* PyTorch. Retrieved September 29, 2020 from https://github.com/pytorch/xla

[20] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs.CV]

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2014. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575 [cs.CV]

[23] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. arXiv:1807.11626 [cs.CV]

[24] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946 [cs.LG]

[25] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2019. EfficientDet: Scalable and Efficient Object Detection. arXiv:1911.09070 [cs.CV]

[26] Kaggle Website. 2020. *iMaterialist (Fashion) 2020 at FGVC7.* "Google AI, CVDF, Fashionpedia and Hearst Magazine". https://www.kaggle.com/c/imaterialist-fashion-2020-fgvc7

[27] Ross Wightman. 2020. *EfficientDet (PyTorch).* rwightman.com. Retrieved September 20, 2020 from https://github.com/rwightman/efficientdet-pytorch

[28] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. 2019. Learning Data Augmentation Strategies for Object Detection. arXiv:1906.11172 [cs.CV]