An Investigation of Transfer Learning Techniques in Instance Segmentation

William Ngo[1] and Andrej Janda[2]

***Abstract*— As performance in instance segmentation has reached remarkable levels, so too has the amount of quality annotated data points required for training. This may not be an issue for large public datasets; it is however an issue when a specific application requires an equally large sized dataset, which due to time and/or cost constraints may be infeasible or impractical. This is why we set out to investigate ways to mitigate these data scarce scenarios by evaluating common transfer learning techniques. As our base network we selected YOLACT++[1] which uses MS COCO [2], a large public dataset of generic common objects, as its source dataset to train on. As our target application, we chose Cityscapes [3] due to its narrow focus and small size. We investigate ways to adjust the network as well as the data. Without any adjustments, we struggle to achieve anywhere near acceptable accuracy. Our optimal configuration requires both data augmentation and transfer learning techniques as our best model is given by having COCO pre-trained weights and allowing for the fine tuning of the entire network on an augmented Cityscapes training dataset that has 1000 COCO images added to it. Code is available on GitHub.[1]**

I. INTRODUCTION

Convolutional neural networks (CNN) have been at the frontier of benchmarks in computer vision tasks ever since AlexNet [4] achieved groundbreaking results on the ILSVRC-2012 competition [5]. The success of CNNs are attributed to their ability to learn rich mid-level representations as opposed to hand-designed low-level features used by classical methods. Although CNNs achieve state-of-the-art results, they have a drawback. CNNs generally can not be trained from scratch as they require a large dataset to train on. This prevents CNNs to be trained directly on problems with limited training data.

To alleviate this issue, transfer learning can typically be used. Transfer learning in the context of CNNs, amounts to taking image representations learned with CNNs on large-scale annotated datasets and transferring them to other vision tasks with limited amounts of training data.

In this project, we wanted to evaluate different transfer learning methods for the task of instance segmentation on 2D images. To perform instance segmentation, YOLACT++ [1] was chosen as our base network. YOLACT++ was chosen as it requires relatively few resources to train and run while achieving results near that of the state-of-the-art. Since we did not have access to a large GPU cluster like in [6], being able to train on a single GPU was essential to the project's success.

[1]MScAc, Department of Computer Science, University of Toronto
[2]MEng, Institute For Aerospace Studies (UTIAS), University of Toronto
[1]https://github.com/ngowilliam1/yolact_cityscapes

The authors of YOLACT++ decided to train the network on the challenging MS COCO dataset [2], a common dataset used to benchmark instance segmentation algorithms. COCO features more than 200k labeled images and has 80 different categories. The Cityscapes dataset [3] was selected as the target dataset for our project. The Cityscapes dataset was chosen as (1) classes within Cityscapes form a subset of the classes in COCO and (2) it is a small dataset of only 3000 annotated images. Both of these reasons indicate that transfer learning between these two datasets is appropriate.

II. RELATED WORKS

A. Instance Segmentation

Large advances in computer vision have rapidly improved object detection and instance segmentation results. In object detection, these advances have been driven by intuitive frameworks such as Faster R-CNN [7] and Fully Convolutional Network (FCN) [8]. Mask-RCNN [6] extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI). In other words, Mask-RCNN is a two-staged approach to instance segmentation. The first stage generates RoIs and the second stage classifies and segments RoIs. The two-staged approach to instance segmentation laid the framework for future similar networks [9] that achieve state-of-the-art results in accuracy. Although two-staged methods achieve state-of-the-art results in accuracy, they require re-pooling of features for each RoI and thus makes them run at sub-real-time speeds ($<$ 30 FPS). Furthermore, as being two-staged and generally larger networks, these approaches require more computational resources compared to one-staged approaches which incentivized us to look into smaller and faster networks for instance segmentation.

The first real-time instance segmentation network that achieved similar results to modern baselines was the base network we selected, YOLACT++. With the advent of YOLACT++, there have been real-time networks that achieve better accurary results than YOLACT++ such as CenterMask [10], SOLOv2 [11] and PolarMask [12]. Despite the increase in accuracy in these networks, we decided to use YOLACT++ as our base network as we found it was better documented and more modular for the different transfer learning scenarios we would test.

B. Transfer Learning

As previously mentioned, transfer learning has the goal of transferring knowledge between related source and target domains. More importantly, they can be used to train networks

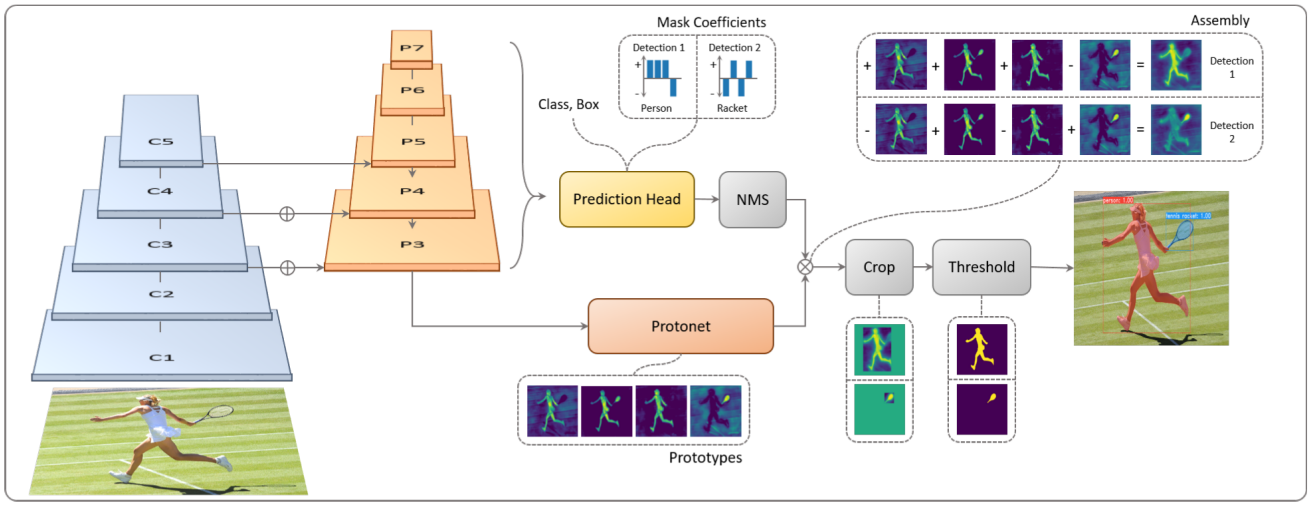


Fig. 1: YOLACT++ Architecture [1]

on a target domain with low amounts of training samples by adapting the classifier trained on other similar data.

At the base level, [13] showed that features from a network trained on ImageNet [5] passed into a linear SVM classifier and subsequently trained on Pascal VOC dataset achieve better results than a network trained from scratch on Pascal VOC. Additionally, [14] showed that results could be further improved by feeding the features into a fully-connected layer and subsequently training the newly added fully-connected layer instead of feeding the features into an SVM classifier. This indicates that a method of transfer learning is that of using a convolutional network as a fixed feature extractor (i.e., removing the last fully-connected layer and attaching a new fully-connected layer, training only that new part and keeping the convolutional layers fixed).

Futhermore, [15] showed that a second method of doing transfer learning is replacing the last fully-connected layer with a new fully-connected layer and allowing the fine-tuning of the convolutional layers. Additionally, [15] showed that depending on the dataset, it may be advantageous to keep some earlier layers fixed and only fine-tune deeper layers of the network instead of fine-tuning all the layers of the CNN. This is motivated by the idea that the earlier features of a CNN contain more generic features like edge detectors that could be useful to different tasks, but further layers of the CNN become progressively more specific to the source task and dataset.

With the observations made by [15], we note that independent of the dataset and task, there is no clear cut winner in which method of transfer learning is best. Thus, in this paper we'll be testing both methods extensively to see which one is best for our task at hand.

Beyond network related techniques, inductive transfer learning focuses more on the datasets. Although the source and target domains are not the same, often times there is some overlap as is illustrated in Figure 2. These techniques employ boosting algorithms like TrAdaBoost [16] to sample

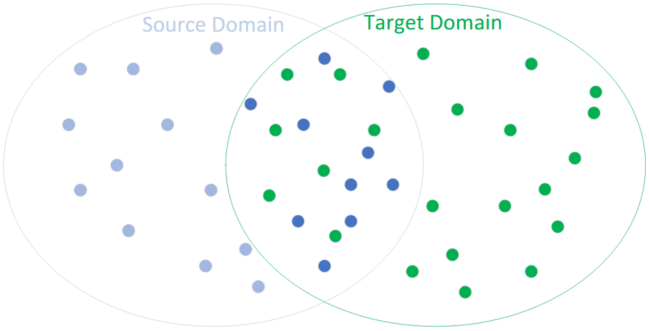


Fig. 2: Domain Overlap [16]

points that exist on this intersection and pick those that are most helpful to learn on the target task. These methods essentially boost the performance of transfer learning methods by augmenting the target dataset with data from the source dataset.

III. DATASETS

A. *COCO*

The authors of YOLACT++ trained their network on MS COCO [2], thus serves as our source dataset for transfer learning. COCO is a large dataset of over 200k labeled images annotated at a per-instance level. Furthermore, COCO is a difficult dataset to perform instance segmentation on as it contains 80 different classes composed of images from complex everyday scenes. Due to the large and difficult properties of the COCO dataset, modern segmentation algorithms such as Mask-RCNN [6] will use COCO as the dataset for their evaluations.

B. *Cityscapes*

As mentioned above, Cityscapes [3] is the target dataset in our study. Cityscapes is a dataset of images of urban street scenes in Germany. It has finely annotated instance-level images for 8 classes. The 8 classes and their distribution

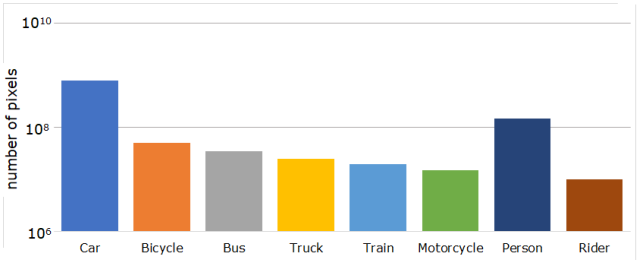


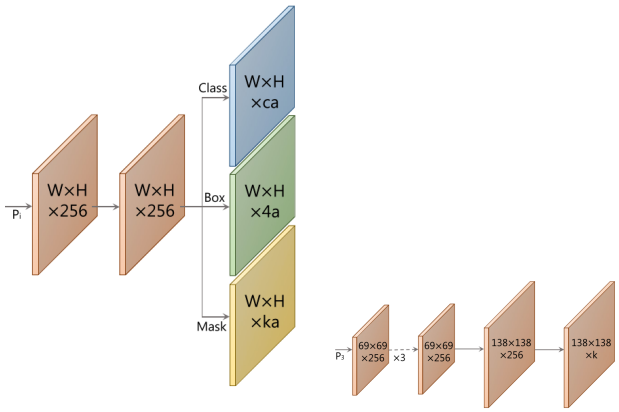
Fig. 3: Number of finely annotated pixels per class

is shown in Figure 3. As noted above, these 8 classes represent a subset of the classes present in COCO and thus indicates that Cityscapes is a good target candidate for transfer learning. Having a source dataset that is similar to the target dataset is critical for the task of transfer learning as pointed out in [16]. The dataset has 3000 training and 500 validation images which is generally not enough to train a network-based object detection framework from scratch hence the need to do transfer learning. In terms of differences that may cause issue, we note that (1) Cityscapes images are wider and larger compared to COCO images and (2) Cityscapes images generally contain more instances in a single image compared to COCO images. This could be problematic as a network trained on COCO may have learned to only expect a few instances per image.

IV. YOLACT++

As previously mentioned, YOLACT++ performs real-time instance segmentation. An overview of the network is shown in Figure 1. The authors of YOLACT++ utilized ResNet-101 [17] with a Feature Pyramid Network (FPN) [18] as their default network backbone with a base image size of 550×550. Each image is preprocessed and converted to a size of 550×550 to be utilized by the network. They also trained the network using the smaller ResNet-50. Using ResNet-50 instead of ResNet-101 resulted in slightly worse mAP results, but as a trade-off had higher FPS and was quicker to train. Most of the models we produced used the ResNet-50 backbone. This was justified as we had limited computational resources and we thought this was appropriate as we are more interested in viewing the performance differences attributed to using the different variants of transfer learning. The ResNet networks were instantiated with pretrained weights from training on ImageNet [5] and when training their network on COCO, they allowed for the fine-tuning of the whole backbone network.

YOLACT++ achieves a considerable speed up compared to two-stage instance segmentation approaches by breaking down the job of creating segmentation masks into two simpler parallel tasks. The first branch produces a set of image-sized prototypes that do not depend on a single instance. The second branch produces the bounding boxes, instance classification and a vector of mask coefficients for each anchor that encodes an instance's representation (see Figure 4a). After these two parallel tasks are complete, for each



(a) Prediction Head [1] (b) Protonet [1]

Fig. 4: Architecture of the two parallel prediction pipelines of YOLACT

instance that survives non-maximum suppression (NMS) on the bounding box, a segmentation mask is constructed by linearly combining the prototype masks with the learned mask coefficients.

To generate the prototype masks, an FCN was implemented whose last layer has k channels which represents the set of k prototype masks for the image. As shown in Figure 4b, a series of deconvolutional layers was used for upsampling to achieve prototype masks of the same height and width as the input image.

To generate the mask coefficients, a third branch was added to the typical anchor-based object detection frameworks that predicts k mask coefficient per anchor, each coefficient would correspond accordingly to each prototype. Thus per anchor, 4 + c + k coefficients are output, 4 for the bounding box, c for the amount of classes and k to represent each mask prototype.

The mask assembly can be implemented efficiently using a matrix multiplication as follows:

$$M = \sigma(PC^T) \tag{1}$$

Where P is an $h\times w\times k$ matrix of prototype masks, C is a $n\times k$ matrix of mask coefficients for the n instances surviving NMS and score thresholding, σ is a sigmoid nonlinearity and M is the resulting instance segmentation masks.

There are several differences between the YOLACT and YOLACT++ network. Differences include adding deformable convolutions into the backbone network, optimizing the prediction layer and adding a fast mask re-scoring branch. A more detailed description of the differences is explained in [1].

V. PROBLEM FORMULATION

As mentioned above, this project aims to evaluate the performance of different transfer learning methods for the task of instance segmentation using YOLACT++ as our base network. We tackled the problem of transfer learning and optimizing the results on Cityscapes by three different

approaches: (1) testing different transfer learning methods, (2) performing appropriate data augmentation, and (3) performing modifications to the network. How we evaluate these methods are discussed in section VI-A.

A. Transfer Learning Methods

As explained in the related works section, there are many different transfer learning methods.

1) Naive Transfer Learning: Class Mappings: The most basic level of transfer learning for our task is to simply feed Cityscapes images into the network trained on COCO. As the classes in Cityscapes form a subset of the classes in COCO this task is simple and requires no training. We simply need to change the output of the results of the network such that if an instance is classified as a class that is not in Cityscapes, then that instance is no longer considered. In other words, if the class prediction output of an instance is not one of the classes present in Figure 3, then that instance is discarded.

2) Necessity of Transfer Learning: Before jumping into our next transfer learning method, we trained the network directly on Cityscapes by only using the ResNet weights trained on ImageNet. In other words, we did not use the weights trained on COCO in this scenario. We did this to motivate the use and necessity of transfer learning as we expected this network to perform poorly due to the small training size of the Cityscapes dataset. As further discussed in the results section, we were correct in assuming that this network would perform poorly.

3) Transfer Learning: Only Fine Tuning The Prediction Head: The second method of transfer learning we employed is by first initializing the network with weights trained on COCO and then perform training on Cityscapes by only allowing the backpropagation of the gradients to flow only in the prediction head. In other words, we only allowed the fine-tuning of the prediction head for this method.

4) Transfer Learning: Fully Fine Tuned Except Backbone: For the third method, we allowed for the fine-tuning of the whole network except for the backbone (ResNet and Feature Pyramid Network). This was motivated by the idea that the earlier features of the network contain more generic features like edge detectors that could be useful to different tasks, but further layers of the network become progressively more specific to the source task and dataset.

5) Transfer Learning: Fully Fine Tuned: Another variation we trained was allowing for the fine-tuning of the entire network. The authors of YOLACT++ [1] allowed for the backpropagation to flow to the entire backbone when they trained their network so we thought it was appropriate to test this variation as well.

As explained in [19], the choice of which of these transfer learning methods to use is governed primarily by two factors, the size of the target dataset and its similarity to the source dataset. For example, if the size of the target dataset is small and it is similar to the source dataset, then fully fine-tuning the network might not be a good idea due to overfitting concerns. In contrast, if the target dataset is large and different from the source dataset, it may be better fully-fine tune the network as it may beneficial to initialize the weights using a pretrained model.

Fig. 5: Can you spot something wrong with this image? Hint: There are no train tracks.

B. Data Augmentation

The biggest challenge with the Cityscapes dataset is its small size as well as its class imbalance. The small size prevents the network from being fully trained as it overfits too readily. The class imbalance means it struggles to classify less frequent class such as riders and motorcycles. Thus to mitigate this, we attempted to increase the size of the dataset in three different ways, each taking a chunk of data from COCO and adding it to the Cityscapes dataset. This is in essence a naive approach to inductive transfer learning where we directly select the classes that we wish to include.

Our first data augmentation approach was to simply take COCO images and annotations containing Cityscapes classes and add them directly to the Cityscapes training set. We added 1000 COCO images excluding those with cars or people as those are readily available in the original dataset. Next, we added 5000 images of all classes.

The second data augmentation approach was to cut and paste COCO instances directly into the Cityscapes images with random, non-overlapping perturbations. We used all classes and selected instances randomly. An example can be found in Figure 5.

C. Network Modifications

1) Focal Loss: A modification to the network that we implemented was the focal loss [20]. Focal loss is typically used in object detection to address class imbalances. Since there are a small number of riders and trains in the Cityscapes training set, we thought that the focal loss was appropriate. Focal loss aims to reshape the cross entropy loss by adding a modulating factor $(1 - p_t)^\gamma$. This modulating factor essentially reduces the loss contribution associated with easy examples and thus puts more emphasis on hard, misclassified examples. As suggested in [20], the α-balanced variant of the focal loss should be used:

$$FL(p_i) = -\alpha(1 - p_i)^\gamma log(p_i)\mathbb{1}_{y=i}$$

where $y \in \{1, 2, ..., |Classes|\}$ is the ground truth label and $p_i \in [0, 1]$ is the model's estimated probability for the class with label $y = i$. During training, we keep the default settings $\alpha_t = 0.25$ and $\gamma = 2$ as suggested by the original paper.

2) Anchor Scale Reduction: Since we noticed that Cityscapes instances were smaller and more numerous in size compared to their COCO counterpart, we thought that reducing the default anchor size of bounding box of the network could help. We thus trained a variation of the network that not only performs transfer learning by fully fine-tuning of the network, but also had the default anchor size of the bounding boxes reduced by a factor of 2.

VI. EXPERIMENTS

A. Evaluation Metrics and Training

We evaluate the effectiveness of the networks by computing the average precision (AP) of the bounding box and the AP of the masks on the Cityscapes validation set (not used during training) as the test set does not have ground truth labels. As we did not have the same computational resources as the authors of YOLACT++, we did not evaluate the frames per second of the network. We train for 60 epochs per model on a NVIDIA GTX TITAN Xp GPU with a minibatch size of 8 images. Training takes approximately 5 hours to train a single variation of the network.

B. Is Transfer Learning Necessary?

Before we even started looking into transfer learning we first wanted to set a benchmark and justify why transfer learning is relevant to this situation. We first ran the naive experiment from sec. V-A.1. These results are shown in the first row of Table I. Next we tried training from scratch as explained in sec. V-A.2. These results are shown in the second row of Table I. Both methods struggle to achieve decent results, meaning (1) models trained on COCO do not generalize to Cityscapes and (2) deep models such as YOLACT++ trained exclusively on Cityscapes overfits on the small training data. This indicates that transfer learning is needed to perform well on this task.

Model	Trained On	Box AP	Mask AP
Yolact	Coco	0.76	14.5
Yolact	Cityscapes	8.13	6.11
Yolact++	Cityscapes	12.89	9.83

TABLE I: Results on Cityscapes Without Transfer Learning

C. Transfer Learning on YOLACT

Our main approach to network related transfer learning was to take pre-trained weights from COCO and train on Cityscapes. We did this in different ways as described in sec. V 3-5. Additionally, we trained a variation of the network in which the default the anchor sizes were halved, as the average size of objects is much smaller than those in the COCO dataset.

Our results for these different experiments can be found in Table II.

The best performance we got on bounding boxes came from training the entire model. Best mask performance was achieved by changing the anchor scales. Its interesting to note that what was most effective for boxes was not as effective for masks.

Experiment	Box AP	Mask AP
Only Prediction Head Tuned	16.68	14.68
Fully Fine Tuned Except Backbone	20.92	17.64
Fully Fine Tuned	**22.43**	14.36
Half Anchor Scales & Fully Fine Tuned	19.33	**19.63**

TABLE II: YOLACT Results

D. Transfer Learning on YOLACT++

We also wanted to see the performance jump between YOLACT and YOLACT++. We initially trained one variation on YOLACT++ to see if there was an major difference, and since the performance difference was quite big, all future models were developed using the YOLACT++ architecture instead of YOLACT. This can be shown by comparing the results between Table II and Table III.

A similar difference between bounding box and mask performance is seen in YOLACT++ performance, shown in Table III. We achieve best bounding box performance when we tune the entire network, including the backbone. Best mask performance is achieved by training only the prediction head.

We believe that learning bounding boxes is harder to overfit than learning segmentation masks as they have fewer parameters. This would explain why only training the prediction head produces best results for the masks while training the entire network produces best results for bounding boxes. Ultimately it all comes down to how far training can go before it starts to overfit.

Seeing as overfitting seems to be the biggest bottle neck, we tried some of the data augmentation methods we referenced in sections V-C.1 & V-B. Focal loss did not seem to add much of an advantage. This could be because we improve accuracy on hard to label instances at the expense of easy to label instances such as cars and people close to the camera.

Using COCO to boost the dataset size however shows to be much more promising and achieves the best overall results. However, we must be careful not to add too many external images as can be seen when we add 5000 images. Since there are only 3000 images in Cityscapes, adding more images from COCO of all classes has detrimental effects. It becomes hard to learn both domains at the same time. However when adding just 1000 images of sparse classes, we see the largest improvement. This is most likely because it improves its performance on classes like motorcycles and riders without hindering the performance on cars and people.

Although pasting COCO object instances directly into Cityscapes images seems to have performed reasonably well at least for bounding boxes, it did not outperform the more naive method. Again, a discrepancy exists between bounding box and segmentation performance.

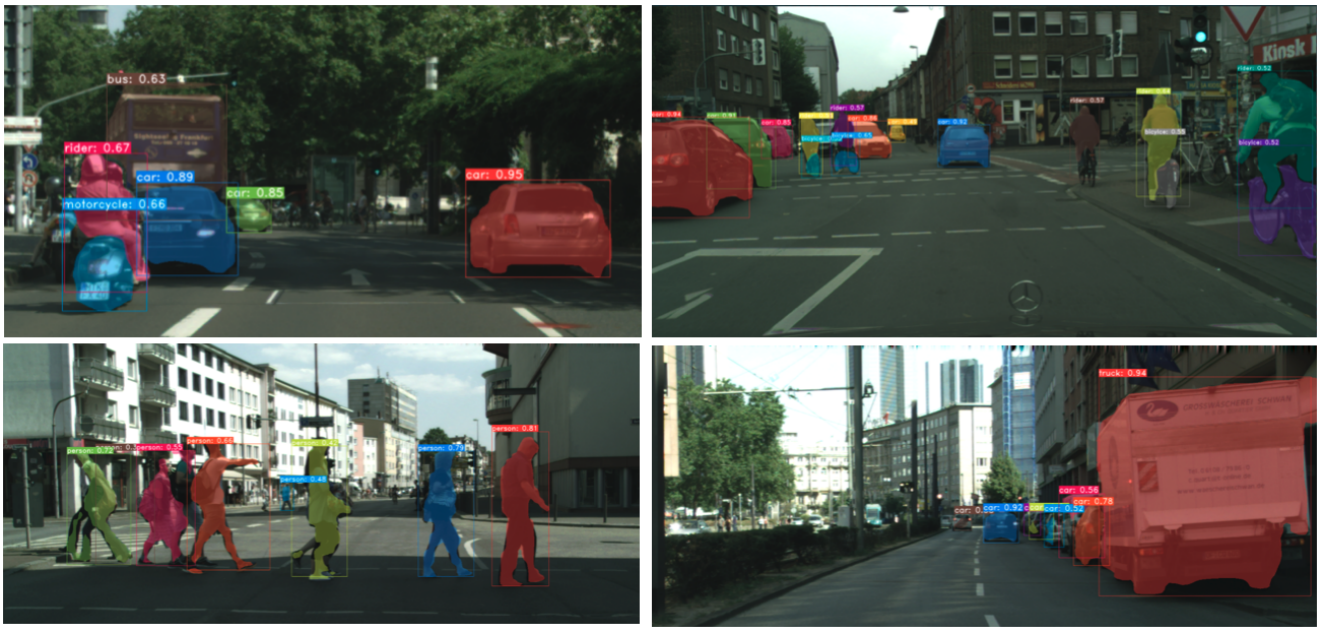


Fig. 6: Cityscapes examples of masks generated using YOLACT++ fully fine tuned using the additional 1000 COCO images

Experiment	Box AP	Mask AP
Only Prediction Head Tuned	23.61	**21.04**
Fully Fine Tuned Except Backbone	24.84	12.08
Fully Fine Tuned	**28.22**	17.59
Add 1000 Coco Images	27.36	**23.02**
Add 5000 Coco Images	1.87	1.5
Add Coco Instances	**28.05**	18.18
Half Anchor Scales	23.76	20.37
Focal Loss	23.13	15.98

TABLE III: YOLACT++ Results

Experiment	Box AP	Mask AP
Only Prediction Head Tuned	23.93	21.60
Fully Fine Tuned Except Backbone	24.52	20.39
Fully Fine Tuned	**27.36**	**23.02**
Half Anchor Scales	23.58	20.57

TABLE IV: YOLACT++ Results for Coco Data Augmentation with additional 1000 images of all classes except cars and people

E. YOLACT++ Coco Data Augmentation

Since the network performed best with this augmented dataset, we decided to perform one last analysis on the different network techniques using a larger dataset. Results can be found in Table IV.

The most surprising result is that because we increased our dataset size, overfitting became harder and we were able to get optimal results for both box and segmentation when we trained the entire network. We note that this method of combining the data augmentation with fully fine tuning the networks yields a good balance of performance for both the Box and Mask AP.

F. Qualitative Results

In addition to the evaluation metrics above, we visually inspected images and found the masks to generally be quite accurate. It was especially accurate for cars and people but also detected motorbikes and riders which before had few representatives in the training set. Examples can be found in Figure 6.

VII. CONCLUSION

In conclusion, we showed that transfer learning is needed to perform well on Cityscapes for the task of instance segmentation and our results are limited primarily by the choice of our base network. We show that the best configuration of the network is to start with coco pre-trained weights and allow for the fine tuning of the entire network on the augmented dataset with the additional 1000 COCO images. It allowed us to train our network further without overfitting and gave us a very good weight initialization. Results are much higher for both bounding box and mask segmentation performance when compared to those without transfer learning. Based off our results, we also believe that the process of learning segmentation masks is easier to overfit compared to learning bounding boxes.

An interesting future work would be to take the weights we trained in this project (trained on COCO and Cityscapes) and perform transfer learning back onto COCO and evaluate the performance on COCO. The idea would be that given Cityscapes images contain generally more instances in a single image compared to COCO, the network would now have learned how to better handle cases in COCO in which there are multiple instances.

REFERENCES

[1] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact++: Better real-time instance segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[2] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, pp. 1097–1105, Curran Associates, Inc., 2012.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015.

[9] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," 2018.

[10] Y. Lee and J. Park, "Centermask : Real-time anchor-free instance segmentation," 2020.

[11] X. Wang, R. Zhang, T. Kong, L. Li, and C. Shen, "Solov2: Dynamic and fast instance segmentation," 2020.

[12] E. Xie, P. Sun, X. Song, W. Wang, D. Liang, C. Shen, and P. Luo, "Polarmask: Single shot instance segmentation with polar representation," 2020.

[13] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," 2014.

[14] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," 2016.

[15] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," 2014.

[16] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," 2018.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[19] F.-F. Li, R. Krishna, and D. Xu, "Transfer learning," 2020. https://cs231n.github.io/transfer-learning/, accessed 2020-09-28.

[20] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2018.