FCHD: A fast and accurate head detector

Aditya Vora, *Johnson Controls Inc.*

***Abstract*—In this paper, we propose FCHD-Fully Convolutional Head Detector, which is an end-to-end trainable head detection model, which runs at 5 fps and with 0.70 average precision (AP), on a very modest GPU. Recent head detection techniques have avoided using anchors as a starting point for detection especially in the cases where the detection has to happen in the wild. The reason is poor performance of anchor-based techniques under scenarios where the object size is small. We argue that a good AP can be obtained with carefully designed anchors, where the anchor design choices are made based on the receptive field size of the hidden layers. Our contribution is two folds. 1) A simple fully convolutional anchor based model which is end-to-end trainable and has a very low inference time. 2) Carefully chosen anchor sizes which play a key role in getting good average precision. Our model achieves comparable results than many other baselines on challenging head detection dataset like BRAINWASH. Along with accuracy, our model has least runtime among all the baselines along with modest hardware requirements which makes it suitable for edge deployments in surveillance applications. The code is made open-source at https://github.com/aditya-vora/FCHD-Fully-Convolutional-Head-Detector.**

***Index Terms*—Head Detection, Crowd Counting, Object Detection, Faster-RCNN.**

I. Introduction

Human head detection is an important problem in computer vision because of its significant applications in the area of surveillance. Crowd counting in surveillance videos can be considered as one of the most important applications of head detection. Several algorithms for crowd counting has been proposed previously, which have achieved pretty good results on very challenging datasets [1], [2], [3]. A very common approach for crowd counting is to train a regression model to predict density maps based on the input image. However, these density map based techniques have several drawbacks. 1) The final crowd count is predicted by taking the sum over the whole density map pixel-wise. While doing this, it does not consider the location on the density map from where the contribution of a count is coming. As a result, false positives occurring in the predictions will contribute to the final crowd count which will lead to erroneous results. 2) The accuracy of the density maps are sensitive to the input image resolution, better results are obtained when the density map resolution is less [4]. Another approach that can solve this problem of crowd counting is a person detection approach, where we can feed the input image directly to a pre-trained model like Faster-RCNN [5] and count the number of bounding boxes in the "person class" in order to get the final crowd count. However, since the performance of Faster-RCNN under highly occluded scenes is a well-known issue, the accuracy of the final crowd count is questionable because, for most of the crowded scenes, humans will be occluded. Moreover, models like Faster-RCNN have very large computational and memory requirements which makes it unfeasible for embedded devices [6]. Thus, deploying such models is practically unfeasible, especially in surveillance applications where "edge-device based applications" are more preferred than cloud-based applications. As a summary, the pros and cons of both the approaches are as follows: 1) Density map based regression techniques have the advantage that they are capable of modeling highly occluded scenes, however, they are prone to high false positive rates and sensitive to input image resolution. 2) Detection based approaches targeting human detection have less false positives but they do not perform well in occluded scenarios, and also they have very huge computational and memory requirements. Thus considering all the pros and cons of the above techniques, a more robust approach for crowd counting is desirable.

In this paper, we try to combine the advantages of both the approaches of crowd counting through a head detection approach. Concretely, we model the head in a scene using a detection approach rather than a regression approach, which will help to eliminate the cons of previous both approaches i.e. less false positives and better performance under occluded scenes (since the head is the most visible part in a scene). Moreover, our model is fully convolutional, which makes it light-weight with better run-time during inference. This is because, fully connected layer ideally consumes most of the memory, as well as inference time as observed in [7]. This makes our model suitable to run on embedded devices which is highly desirable for surveillance applications. Unlike previous approaches [8], [9] we propose an anchor based fully convolutional, and end-to-end trainable model for head detection. Our model is based upon Faster-RCNN [5], however, unlike a two-stage pipeline in Faster-RCNN (Region Proposal Network + Classification), we have a single stage pipeline which can directly perform the head detection. Moreover, unlike randomly chosen anchor scales, we carefully select anchors based on the effective receptive field size of the network [10]. This gives a major performance boost in the final average precision as shown by our ablation experiments. Our contributions are as follows:

- An anchor based, fully convolutional, end-to-end trainable head detection model, where receptive field size based anchor scales plays a vital role in getting a significant performance boost.
- A head detection model is proposed, which has a good inference time, as well as low memory requirements.
- We achieve comparable results to some baseline techniques along with outperforming some of the other previous approaches for head detection on challenging dataset like BRAINWASH dataset [8].

II. Related Work

A brief review of previous attempts for solving the crowd counting and head detection is explained below.

Crowd counting approaches are mainly based on regression-based techniques, where several deep learning as well as feature engineering based techniques have been proposed in the past [11], [12], [13], [14]. Chan et. al. [1] proposed a regression-based technique where hand-crafted features like edges and textures are extracted from the motion segmented region in a video. After feature extraction, in order to map the extracted features to actual crowd count, a linear regression model is trained. However, the features used in this technique is highly sensitive to the segmentation area, the mapping that is learned using a regression model will vary with different camera locations. Because of this reason, this technique fails to generalize to unseen scenarios. Zhang et. al. [3] tried to address this problem using a deep learning approach, where they trained a CNN in order to predict the density maps as well as the crowd count. They eliminated the drawbacks of generalization that the previous crowd counting techniques faced, with a data-driven approach. However, this method requires a perspective map in order to normalize the scales of the heads in an image. Since in most of the practical scenarios perspective maps are not readily available, which restricts its applicability. This problem was tackled with a multi-column architecture [2]. In this architecture, a CNN with 3 columns is used, and the convolutional kernel sizes are fixed in such a way that they are capable of modeling various scales of heads. This helps to get rid of the perspective normalization map. This makes the algorithm more suitable for practical scenarios. However, since it predicts density maps it has the following drawbacks as discussed in the previous section.

Many previous head detection approaches for crowd counting has been published which combines the positive points of all the previous techniques in a head detection approach [15]. [16] proposed a head detection approach for crowd counting where the detector is trained using a state-of-the-art cascade of boosted integral features. However, because of the usage of handcrafted features, it's performance can severely get affected under high scene and scale variations. Vu et. al. [9] proposed a technique for head detection, where two CNN models were used to perform head detections, the first model was used to predict the scales and the positions of the head directly from the image. While the second model is used for modeling the pairwise relationships among the objects. Several proposal free approaches have also been proposed [8]. In this approach, head detections are produced from the CNN encoders using a regression module. This regression is generally composed of LSTM so that the variable length output prediction is possible.

III. Method

A. Model architecture

Deep architectures like [17], [18], [19], [20] are capable of learning "generalized features", which are useful for variety of tasks. We start off by leveraging one of such already existing pre-trained models VGG16 [19], as our base model for our architecture. We remove the final layers succeeding the conv5 layer, and use the remaining weights as a starting point for our new training. The architecture is shown in Fig. 1. 3 new convolutional layers are added: (1) The first convolutional layer is like any other convolutional layer and its purpose is to encode the information obtained from the previous layers. (2) The second convolutional layer is the regression head responsible for predicting localization coordinates. (3) The third convolutional layer is the classification head which predicts probability scores of a head. The second and third convolutional layers are implemented using a 1×1 convolutional kernel. The outputs from the regression head undergo bounding box transformation, where the predictions are converted to spatial coordinates. The number of kernels in both classification and regression heads depends on a number of anchors at each pixel location (N) as shown in Fig. 1.

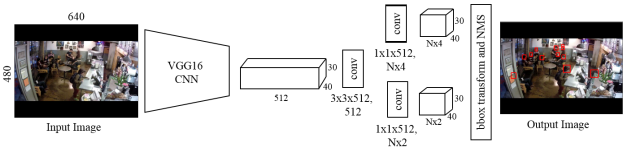

Fig. 1. Architecture of our head detection model. After generating a set of predefined anchors, our model uses a fully convolutional network to estimate the coordinates from the anchors alongwith the probability score. There are two heads to the model, 1) Regression head, $40 \times 30 \times (N \times 4)$. 2) Classification head, $40 \times 30 \times (N \times 2)$. Each $1 \times 1 \times (N \times 4)$ output from the regression head will indicate the scale and shift of N anchors at that location in the feature map. Here, $N = 2$ which is the number of anchors.

B. Design of Anchor Scales

Anchors are a predefined set of bounding boxes with respect to which the scales and shifts are predicted in an object recognition system [5]. These anchors were first introduced in Faster-RCNN. However, unlike Faster-RCNN which uses 9 anchors at each pixel location with aspect ratio of $0.5, 1$ and 2 and anchor sizes of 128×128, 256×256 and 512×512, we use anchors with only one aspect ratio i.e. $1:1$ and 2 sizes i.e. 32×32 and 64×64. The selected aspect ratio of anchors is square since it will match the aspect ratio of heads.

The anchor scales are selected based on the "effective receptive field" rather than the "theoretical receptive field". As pointed out in [10], a unit in a CNN has two types of receptive field, one is a theoretical receptive field and the other is the effective receptive field. Not all pixels of theoretical receptive field contribute to the final output. Only a few center pixels have a much larger effect on final output compared to outer pixels of the receptive field. This center region is called effective receptive field. In the conv5 layer of VGG16, the theoretical receptive field size is 228 pixels. Taking note of this receptive field, we prefer anchors of scales 2 and 4 which leads to anchor sizes 32×32 and 64×64 respectively. Anchor size is computed as $feat\_stride \times aspect\_ratio \times anchor\_scale$ where the $feat\_stride$ of the conv5 layer is 16. As anchor sizes are approximately scaled down by 3.5 times the theoretical receptive field, it has good chance to match the effective receptive field. Another intuitive reason for selecting the anchor scales of size 2 and 4 can be given from Fig. 2. The receptive field size for the conv5 layer is shown in red, the anchor scales 8, 16, 32 w.r.t Faster-RCNN is shown in green (only for one aspect ratio), and the anchor scales

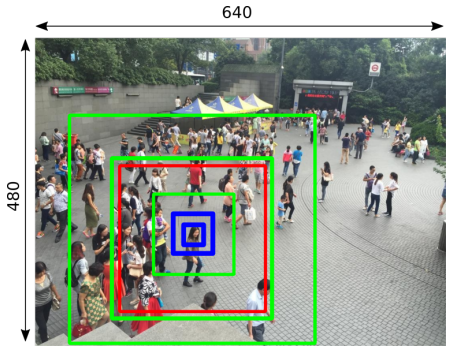

Fig. 2. Anchor scales of Faster-RCNN vs. Ours. Red Box: Receptive field of the conv5 layer on the input image (228×228). Green Boxes: Anchor sizes used in Faster-RCNN i.e. 128×128, 256×256 and 512×512 respectively. Blue Boxes: Anchor sizes used in our model i.e. 32×32 and 64×64. It can be observed that blue boxes have a better chance for giving accurate prediction of head locations compared to the green ones.

used by us i.e. 2, 4 are shown in blue. Faster-RCNN deals with localizing mainly large objects in a scene and thus large anchor scales would be a good starting point for the network to make predictions. However, in our scenario the heads are generally very small in size in an image, thus selecting large anchor scales does not make sense because in that case, the network has to predict very large scales and shifts for accurate localization. Thus by ensuring that the anchor scales are small, we make sure that the network has a good probability of accurately localizing the head from the pre-defined anchor location. This can be seen from the Fig. 2. Designing anchor scales proved to be the most important step in getting a good average precision as it will be seen in our experiments.

C. Training

Dataset: Our model is trained on 10461 training images from the BRAINWASH dataset [8][2]. Each image is of size 640×480. Every image undergoes preprocessing, where along with mean subtraction and normalization of the image, it also undergoes scaling. The smaller side of the image is scaled to 600 pixels if the dimension exceeds 600 pixels. We validate the model after each epoch with the validation split of the same dataset which contains 493 images.

Approach: Using our designed set of anchor scales, we generate a set of predefined anchors on the original image. The anchors are defined at a stride of 16 since each pixel shift in the conv5 layer will have a 16-pixel shift in the input layer (because of 4 pooling layers). We select two anchors/pixel of size 32×32 and 64×64. The input resolution of the image is 640×480 and thus the feature map obtained at the conv5 layer is of size 40×30. Thus for each image we obtain 2400 anchors ($40 \times 30 \times 2$). For each anchor, the network will predict 4 regression coordinates in the form of scales and shifts the anchors need to undergo to accurately localize the head, and 2 softmax outputs for head detection score. This functionality can be naturally implemented using a 1×1 convolutional layer which can be seen from our regression and classification heads from Fig. 1. We do not consider the anchors going out of the boundary of the image for training. In order to assign binary labels to the anchors for training, we follow these 2 strategies: (1) The anchor that have IoU ≥ 0.7 with ground-truth is labeled positive. (2) The anchors with maximum IoU with the ground-truth is assigned a positive label. We follow both strategies since in some cases the first condition might fail to assign a positive label to any of the anchors. Negative labels are assigned to those anchors which have IoU ≤ 0.3. The anchors that do not satisfy both the conditions are don't care and are not included in the training. We process a single image at a time during training. And from the anchors with assigned labels, we pick a total of 32 positive and negative anchors in a ratio of $1:1$.

Loss function: The loss function used for the training of the model is a multi-task loss function, similar to that defined in training of an RPN in [5]. It is shown in Eq. 1.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{1}$$

where i is the index of chosen anchor which will range through 32 selected anchors. p_i is the predicted probability that an anchor i contains head and p_i^* is the ground-truth label i.e. 1 or 0. t_i is the parameterized coordinates of the predicted bounding box (i.e. scale and shift), and t_i^* is the parameterized coordinates of the ground truth. This parameterized coordinates are defined in the same manner as in [7]. L_{cls} denotes the classification loss, whereas L_{reg} denotes the regression loss. L_{cls} is computed over all the anchors whereas L_{reg} is computed over only positive anchors and this is taken care by p_i^* in front of L_{reg} in the equation. L_{cls} is the softmax loss over the two classes (head vs. background). Whereas the L_{reg} is the smooth L_1 loss as defined in [7]. Both the loss terms are normalized by N_{cls} and N_{reg} which are the number of samples accounted for classification and regression respectively.

Hyperparameters: The base model is initialized with pre-trained VGG16, which is trained using ImageNet dataset [21]. All the layers of the pre-trained model and the new layers are retrained. The new layers are initialized with random weights sampled from a standard normal distribution with $\mu = 0$ and $\sigma = 0.01$. The weight decay during the training is 0.0005. We fine-tune the entire model with SGD. The learning rate for training is set to 0.001, and the model is trained for 15 epochs which are close to $160k$ iterations. We decay the learning rate with a scale of 0.1 after completing 8 epochs. Our method is implemented with PyTorch [22].

IV. Experiments

We test our model on the test split of the BRAINWASH dataset containing 484 images. Following settings are kept constant throughout the experiment. Anchor scales are set to 2 and 4. Ablation studies with varying the anchor scales are performed. Non Maximum suppression (NMS) is performed during the test phase and not in the training phase. The threshold for NMS is set to 0.3. NMS is a kind of post-processing technique to filter out highly overlapping detections. In this

[2]The BRAINWASH dataset is available open-source for download from https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/software-and-datasets

TABLE I
AVERAGE PRECISION OF OUR MODEL COMPARED TO OTHERS

Method	AP
Overfeat - AlexNet	0.62
ReInspect, L_{fix}	0.60
ReInspect, L_{firstk}	0.63
ReInspect, $L_{hungarian}$	0.78
Ours	**0.70**

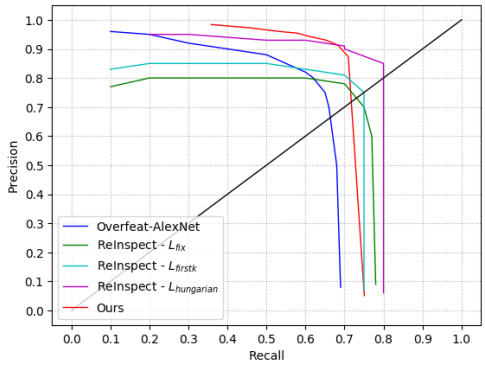

Fig. 3. Precision-Recall analysis of our technique compared to baselines.

technique, any two detections are suppressed to one if the IoU among them is greater than a threshold. Following evaluation metrics is used, i.e. head is considered to be correctly detected if the IoU of the predicted bounding box with the ground truth is ≥ 0.5 [23]. In order to quantify the performance of our system, we plot the precision-recall curve with the baselines.

Baselines: The baseline techniques are the same which were considered for comparison in [8]. The first baseline technique is the OverFeat with AlexNet architecture [24]. The second baseline is the current state of the art technique called ReInspect [8]. We prefer these baselines since they are all benchmarked on the most recent and challenging BRAINWASH dataset. There are 3 variants of ReInspect, which uses various types of losses i.e. L_{fix}, L_{first}, $L_{hungarian}$. We compare our technique with all of these baselines.

Performance evaluation: We compare the precision-recall curve of our technique with other baselines. The result is shown in the Fig. 3. As it can be seen from the figure, the precision-recall obtained by our technique is better than 3 of the baselines i.e. Overfeat-AlexNet, ReInspect, L_{fix} and ReInspect, L_{firstk}. Although the precision-recall curve is below the fourth baseline, which is the best performing model at EER (equal error rate), our model gives best performance when the recall is compromised a bit i.e. best precision of 0.92 at a recall of 0.65 which can be observed from the precision-recall curve. One reason why we obtain lower recall is that many heads appearing in the image are very small in size which the network sometimes fails to detect because of large receptive field size of 228. This can be improved if we consider anchors from previous layers as well for detection where the receptive field size is less. The average precision comparison of our technique with the baseline is shown in Table I. As it can be seen in the table that AP precision obtained by our model is 0.70, which is better compared to three of the baselines whereas comparable to the best performing model.

TABLE II
RUN-TIME COMPARISON AT 640×480 RESOLUTION

Platform	Model	Speed
Quadro M1000M	ReInspect	1fps
	FCHD	5fps
Jetson TX2	ReInspect	NA
	FCHD	1.6fps

TABLE III
EFFECT ON AP WITH VARIOUS ANCHOR SIZES

AP with various Anchor Sizes	
Anchor Size	**AP**
32×32, 64×64	0.70
64×64, 128×128	0.53
128×128, 256×256	0.45

Timings: We perform a timing comparison of our technique with the baselines. The timings of our model are benchmarked on two platforms. The first is a general purpose GPU platform where we use NVidia Quadro M1000M GPU which has 512 CUDA cores. The second platform is NVidia Jetson TX2 which has 256 CUDA cores. Since Jetson is the most suitable GPU device for edge deployments we prefer to benchmark timings on this platform as well. The inference time for all the test images i.e. 484 images are computed and the final timings are computed using the average of inference time of all test images. The timings are shown in Table. II. As it can be seen from the table that our model runs at 5fps which is $5\times$ faster than ReInspect on the same platform. Moreover, our model runs at 1.6fps on Jetson TX2 embedded platform whereas the other baseline model fails to load because of memory requirements. This is a major advantage of our model as along with being accurate, it is also suitable for edge deployments.

Ablation Experiments: In order to measure the effect of anchor size in the final accuracy (AP), we perform an ablation experiment where we train 3 different models with varying anchor sizes i.e. (1) 32×32, 64×64 (default setting), (2) 64×64, 128×128, (3) 128×128, 256×256. All the models are trained with the same hyperparameter settings. We see from Table. III that we achieve best performance when the anchor sizes are 32×32, 64×64 (equal to the effective receptive field) which justifies our claim made in previous sections.

V. CONCLUSION

In this work, we propose an anchor based fully convolutional end-to-end trainable model for detecting heads in complex scenes. Our model achieves 0.70 average precision on challenging head detection dataset like BRAINWASH, which is comparable to previous techniques. Along with being accurate, it has very less run-time i.e. 5fps on Quadro and 1.6fps on Jetson TX2. In future, we plan to extend this work by using the anchors of various scales from different convolutional layers and combine the detections in order to get better overall accuracy especially in cases where the head sizes are small.

REFERENCES

[1] A. B. Chan, Z.-S. J. Liang, and N. Vasconcelos, "Privacy preserving crowd monitoring: Counting people without people models or tracking," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–7.

[2] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-image crowd counting via multi-column convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 589–597.

[3] C. Zhang, H. Li, X. Wang, and X. Yang, "Cross-scene crowd counting via deep convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 833–841.

[4] D. Kang, Z. Ma, and A. B. Chan, "Beyond counting: comparisons of density maps for crowd analysis tasks-counting, detection, and tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1137–1149, 2017.

[6] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE CVPR*, vol. 4, 2017.

[7] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[8] R. Stewart, M. Andriluka, and A. Y. Ng, "End-to-end people detection in crowded scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2325–2333.

[9] T.-H. Vu, A. Osokin, and I. Laptev, "Context-aware cnns for person head detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2893–2901.

[10] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in neural information processing systems*, 2016, pp. 4898–4906.

[11] D. Kong, D. Gray, and H. Tao, "A viewpoint invariant approach for crowd counting," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3. IEEE, 2006, pp. 1187–1190.

[12] C. Shang, H. Ai, and B. Bai, "End-to-end crowd counting via joint learning local and global count," in *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1215–1219.

[13] L. Boominathan, S. S. Kruthiventi, and R. V. Babu, "Crowdnet: A deep convolutional network for dense crowd counting," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 640–644.

[14] D. B. Sam, S. Surya, and R. V. Babu, "Switching convolutional neural network for crowd counting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, no. 3, 2017, p. 6.

[15] D. Merad, K.-E. Aziz, and N. Thome, "Fast people counting using head detection from skeleton graph," in *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*. IEEE, 2010, pp. 233–240.

[16] V. B. Subburaman, A. Descamps, and C. Carincotte, "Counting people in the crowd using a generic head detector," in *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*. IEEE, 2012, pp. 470–475.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[23] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.