PELEE: A REAL-TIME OBJECT DETECTION SYSTEM ON MOBILE DEVICES

Robert J. Wang, Xiang Li, Shuang Ao & Charles X. Ling
Department of Computer Science
University of Western Ontario
London, Ontario, Canada, N6A 3K7
{jwan563,lxiang2,sao,charles.ling}@uwo.ca

ABSTRACT

An increasing need of running Convolutional Neural Network (CNN) models on mobile devices with limited computing power and memory resource encourages studies on efficient model design. A number of efficient architectures have been proposed in recent years, for example, MobileNet, ShuffleNet, and NASNet-A. However, all these models are heavily dependent on depthwise separable convolution which lacks efficient implementation in most deep learning frameworks. In this study, we propose an efficient architecture named PeleeNet, which is built with conventional convolution instead. On ImageNet ILSVRC 2012 dataset, our proposed PeleeNet achieves a higher accuracy by 0.6% (71.3% vs. 70.7%) and 11% lower computational cost than MobileNet, the state-of-the-art efficient architecture. Meanwhile, PeleeNet is only 66% of the model size of MobileNet. We then propose a real-time object detection system by combining PeleeNet with Single Shot MultiBox Detector (SSD) method and optimizing the architecture for fast speed. Our proposed detection system[1], named Pelee, achieves 76.4% mAP (mean average precision) on PASCAL VOC2007 and 22.4 mAP on MS COCO dataset at the speed of 17.1 FPS on iPhone 6s and 23.6 FPS on iPhone 8. The result on COCO outperforms YOLOv2 in consideration of a higher precision, 13.6 times lower computational cost and 11.3 times smaller model size.

1 INTRODUCTION

There has been a rising interest in running high-quality CNN models under strict constraints on memory and computational budget. Many innovative architectures, such as MobileNets (Howard et al. (2017)), ShuffleNet (Zhang et al. (2017)), NASNet-A (Zoph et al. (2017)), have been proposed in recent years. However, all these architectures are heavily dependent on depthwise separable convolution (Szegedy et al. (2015)) which lacks efficient implementation. Meanwhile, there are few studies that combine efficient models with fast object detection algorithms (Huang et al. (2016b)). This research tries to explore the design of an efficient CNN architecture for both image classification tasks and object detection tasks. It has made a number of major contributions listed as follows:

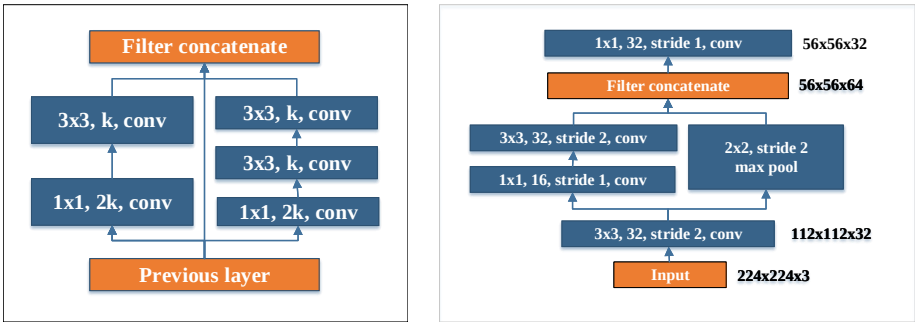
We propose a variant of DenseNet (Huang et al. (2016a)) architecture called PeleeNet for mobile devices. PeleeNet follows the innovate connectivity pattern and some of key design principals of DenseNet. It is also designed to meet strict constraints on memory and computational budget. Experimental results on Stanford Dogs (Khosla et al. (2011)) dataset show that our proposed PeleeNet is higher in accuracy than the one built with the original DenseNet architecture by 5.05% and higher than MobileNet (Howard et al. (2017)) by 6.53%. PeleeNet achieves a compelling result on ImageNet ILSVRC 2012 (Deng et al. (2009)) as well. The top-1 accuracy of PeleeNet is 71.3% which is higher than that of MobileNet by 0.6%. It is also important to point out that PeleeNet is only 66% of the model size of MobileNet. Some of the key features of PeleeNet are:

- **Two-Way Dense Layer** Motivated by GoogLeNet (Szegedy et al. (2015)), we use a 2-way dense layer to get different scales of receptive fields. One way of the layer uses a small

[1]The code and models are available at: https://github.com/Robert-JunWang/Pelee

kernel size (3x3), which is good enough to capture small-size objects. The other way of the layer uses two stacked 3x3 convolution to learn visual patterns for large objects. The structure is shown on Fig. 1.a,

- **Stem Block** Motivated by Inception-v4 (Szegedy et al. (2017)) and DSOD (Shen et al. (2017)), we design a cost efficient stem block before the first dense layer. The structure of stem block is shown on Fig. 1.b. This stem block can effectively improve the feature expression ability without adding computational cost too much - better than other more expensive methods, e.g., increasing channels of the first convolution layer or increasing growth rate.



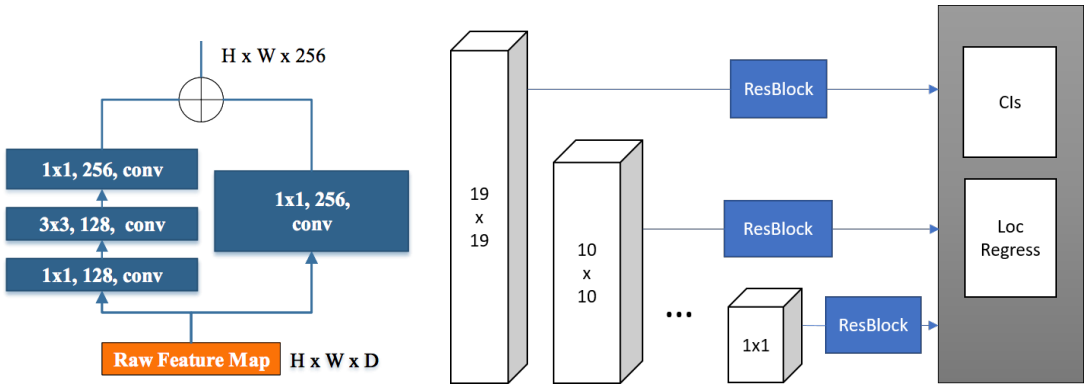
(a) 2-way dense layer (b) Stem block

Figure 1: Structure of 2-way dense layer and stem block

- **Dynamic Number of Channels in Bottleneck Layer** Another highlight is that the number of channels in the bottleneck layer varies according to the input shape to make sure the number of output channels does not exceed the number of its input channels. Compared to the original DenseNet structure, our experiments show that this method can save up to 28.5% of the computational cost with a small impact on accuracy.
- **Transition Layer without Compression** Our experiments show that the compression factor proposed by DenseNet hurts the feature expression. We always keep the number of output channels the same as the number of input channels in transition layers.
- **Composite Function** To improve actual speed, we use the conventional wisdom of post-activation (Convolution - Batch Normalization (Ioffe & Szegedy (2015)) - Relu) as our composite function instead of pre-activation used in DenseNet. For post-activation, all batch normalization layers can be merged with convolution layer at the inference stage, which can accelerate the speed greatly. To compensate for the negative impact on accuracy caused by this change, we use a shallow and wide network structure. We also add a 1x1 convolution layer after the last dense block to get the stronger representational abilities.

We optimize the network architecture of Single Shot MultiBox Detector (SSD) (Liu et al. (2016)) for speed acceleration and then combine it with PeleeNet. Our proposed system, named Pelee, achieves 76.4% mAP on PASCAL VOC (Everingham et al. (2010)) 2007 and 22.4 mAP on COCO. It outperforms YOLOv2 (Redmon & Farhadi (2016)) in terms of accuracy, speed and model size. The major enhancements proposed to balance speed and accuracy are:

- **Feature Map Selection** We build object detection network in a way different from the original SSD with a carefully selected set of 5 scale feature maps (19 x 19, 10 x 10, 5 x 5, 3 x 3, and 1 x 1). To reduce computational cost, we do not use 38 x 38 feature map.
- **Residual Prediction Block** We follow the design ideas proposed by Lee et al. (2017) that encourage features to be passed along the feature extraction network. For each feature map used for detection, we build a residual (He et al. (2016)) block (ResBlock) before conducting prediction. The structure of ResBlock is shown on Fig. 2
- **Small Convolutional Kernel for Prediction** Residual prediction block makes it possible for us to apply 1x1 convolutional kernels to predict category scores and box offsets. Our experiments show that the accuracy of the model using 1x1 kernels is almost the same as



(a) ResBlock (b) Network of Pelee

Figure 2: Residual Prediction Block

that of the model using 3x3 kernels. However, 1x1 kernels reduce the computational cost by 21.5%.

We provide an efficient implementation of SSD algorithm on iOS. We have successfully ported SSD to iOS and provided an optimized code implementation. Our proposed system runs at the speed of 17.1 FPS on iPhone 6s and 23.6 FPS on iPhone 8. The speed on iPhone 6s, a phone released in 2015, is 2.6 times faster than that of the official SSD implementation on a server with a powerful Intel i7-6700K@4.00GHz CPU.

2 Architecture of PeleeNet

Our proposed architecture is shown as follows in Table 1. The entire network consists of a stem block and four stages of feature extractor. Except the last stage, the last layer in each stage is average pooling layer with stride 2. A four-stage structure is a commonly used structure in the large model design. ShuffleNet (Zhang et al. (2017)) uses a three stage structure and shrinks the feature map size at the beginning of each stage. Although this can effectively reduce computational cost, we argue that early stage features are very important for vision tasks, and that premature reducing the feature map size can impair representational abilities. Therefore, we still maintain a four-stage structure. The number of layers in the first two stages are specifically controlled to an acceptable range.

3 Experimental results

3.1 Ablation Study

3.1.1 Dataset

We build a customized Stanford Dogs dataset for ablation study. Stanford Dogs (Khosla et al. (2011)) dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image classification. We believe the dataset used for this kind of task is complicated enough to evaluate the performance of the network architecture. However, there are only 14,580 training images, with about 120 images per class, in the original Stanford Dogs dataset, which is not large enough to train the model from scratch. Instead of using the original Stanford Dogs, we build a subset of ILSVRC 2012 according to the ImageNet wnid used in Stanford Dogs. Both training data and validation data are exactly copied from the ILSVRC 2012 dataset. In the following chapters, the term of Stanford Dogs means this subset of ILSVRC 2012 instead of the original one. Contents of this dataset:

- Number of categories: 120
- Number of training images: 150,466

Stage		Layer	Output Shape
Input			224 x 224 x 3
Stage 0	Stem Block		56 x 56 x 32
Stage 1	Dense Block	DenseLayer **x 3**	28 x 28 x 128
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 2	Dense Block	DenseLayer **x 4**	14 x 14 x 256
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 3	Dense Block	DenseLayer **x 8**	7 x 7 x 512
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 4	Dense Block	DenseLayer **x 6**	7 x 7 x 704
	Transition Layer	1 x 1 conv, stride 1	
Classification Layer		7 x 7 global average pool	1 x 1 x 704
		1000D fully-connecte,softmax	

Table 1: Overview of PeleeNet architecture

- Number of validation images: 6,000

3.1.2 Effects of Various Design Choices on the Performance

We build a DenseNet-like network called DenseNet-41 as our baseline model. There are two differences between this model and the original DenseNet. The first one is the parameters of the first conv layer. There are 24 channels on the first conv layer instead of 64, the kernel size is changed from 7 x 7 to 3 x 3 as well. The second one is that the number of layers in each dense block is adjusted to meet the computational budget.

All our models in this section are trained by PyTorch with mini-batch size 256 for 120 epochs. We follow most of the training settings and hyper-parameters used in ResNet on ILSVRC 2012. Table 2 shows the effects of various design choices on the performance. We can see that, after combining all these design choices, PeleeNet achieves 79.25% accuracy on Stanford Dogs, which is higher in accuracy by 4.23% than DenseNet-41 at less computational cost.

Table 2: Effects of various design choices and components on performance

	From DenseNet-41 to PeleeNet						
Transition layer without compression		✓	✓	✓	✓	✓	✓
Post-activation			✓			✓	✓
Dynamic bottleneck channels				✓	✓	✓	✓
Stem Block					✓	✓	✓
Two-way dense layer						✓	✓
Go deeper (add 3 extra dense layers)							✓
Top 1 accuracy	**75.02**	76.1	75.2	75.8	76.8	78.8	**79.25**

3.1.3 Comparison with MobileNets and Other Models

This section describes the result on Stanford Dogs and the result compared to other pre-trained models. Since both training data and validation data of our customized Stanford Dogs are from counterparts in ILSVRC 2012 without any change, we can evaluate a model pre-trained on ILSVRC 2012 on this dataset. By this way, we can get some baseline information to help evaluate our model design. We have evaluated some pre-trained models, e.g. MobileNet, DenseNet121, VGG16 and

ResNet50. The top-1 accuracy of the pre-trained MobileNet is 73.5%, which is slightly higher than the MobileNet model we trained from scratch (72.9%) on this dataset.

We use a different data augmentation method in this section. Besides random-sized cropping, we also randomly adjust brightness and contrast of training images. This new data augmentation approach brings a 0.3% performance boost. Different from previous sections, the model is trained with a cosine learning rate annealing schedule, similar to what is used by Pleiss et al. (2017) and Loshchilov & Hutter (2016).

Cosine Learning Rate Annealing means that the learning rate decays with a cosine shape (the learning rate of epoch t ($t <= 120$) set to $0.5 * lr * (\cos(\pi * t/120) + 1)$.

As we can see from Table 3, PeleeNet achieves a compelling result on Stanford Dogs dataset. The top 1 accuracy is 80.03%, which is 6.53% higher than that of MobileNet. This accuracy is even higher than that of DenseNet-121 and ResNet50. Moreover, the computational cost of PeleeNet is only 18.6% of the cost of DenseNet-121 and only 13.7% of the one in ResNet50.

Table 3: **Results on Stanford Dogs.** MACs: the number of Multiply-Accumulates which measures the number of fused Multiplication and Addition operations[3]

Model	Computational Cost (Million MACs)	Model Size (Million Parameters)	Top-1 Accuracy (%)
VGG16	15,346	138	75.45
ResNet50	3,832	23.48	79.48
DenseNet-121	2,833	6.87	78.65
1.0 MobileNet	569	3.32	73.5
DenseNet-41	545	1.13	75.02
PeleeNet	507	2.18	**80.03**

3.2 Results on ImageNet ILSVRC 2012

Our PeleeNet is trained by PyTorch with mini-batch size 256 for 120 epochs. The model is trained with a cosine learning rate annealing schedule which starts from 0.1. Other hyper-parameters are the same as the one used on Stanford Dogs dataset. As can be seen from Table 4, PeleeNet achieves a higher accuracy than that of MobileNet and ShuffleNet at no more than 66% model size and the lower computational cost. The model size of PeleeNet is only 1/49 of VGG16.

Table 4: Results on ImageNet ILSVRC 2012

Model	Computational Cost (Million MACs)	Model Size (Million Parameters)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
VGG16	15,346	138	71.5	89.8
1.0 MobileNet	569	4.24	70.7	89.5
ShuffleNet 2x (g = 3)	524	5.2	70.9	-
NASNet-A	564	5.3	74.0	91.6
PeleeNet (ours)	508	2.8	**71.3**	**90.3**

3.3 Results on VOC 2007

Our object detection system is based on the source code of SSD[4] and is trained with Caffe (Jia et al. (2014)). The batch size is set to 32. The learning rate is set to 0.005 initially, then it decreased by a factor of 10 at 80k and 100k iterations,respectively. The total iterations are 120K.

[3]From https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

[4]https://github.com/weiliu89/caffe/tree/ssd

3.3.1 Effects of Various Design Choices

Table 5 shows the effects of our design choices on performance. We can see that residual prediction block can effectively improve the accuracy. The model with residual prediction block achieves a higher accuracy by 2.2% than the model without residual prediction block. The accuracy of the model using 1x1 kernels for prediction is almost same as the one of the model using 3x3 kernels. However, 1x1 kernels reduce the computational cost by 21.5% and the model size by 33.9%.

Table 5: Effects of various design choices on performance

38x38 Feature	ResBlock	Kernel Size for Prediction	Computational Cost (Million MACs)	Model Size (Million Parameters)	mAP (%)
✓	✗	3x3	1,670	5.69	69.3
✗	✗	3x3	1,340	5.63	68.6
✗	✓	3x3	1,470	7.27	70.8
✗	✓	1x1	1,210	5.43	**70.9**

3.3.2 Comparison with Other Frameworks

As can be seen from Table 6, the accuracy of Pelee is higher than that of TinyYOLOv2 by 13.8% and higher than that of SSD+MobileNet (Huang et al. (2016b)) by 2.9%. It is even higher than that of YOLOv2-288 at only 14.5% of the computational cost of YOLOv2-288. Pelee achieves 76.4% mAP when we take the model trained on COCO trainval35k as described in Section 3.4 and fine-tuning it on the 07+12 dataset.

Table 6: **Results on PASCAL VOC 2007.** Data: 07+12: union of VOC2007 and VOC2012 trainval. 07+12+COCO: first train on COCO trainval35k then fine-tune on 07+12

Model	Computational Cost (Million MACs)	Model Size (Million Parameters)	Data	mAP (%)
YOLOv2-288	8,360	67.13	07+12	69.0
Tiny-YOLOv2	3,490	15.86	07+12	57.1
SSD+MobileNet	1,150	5.77	07+12	68
Pelee (ours)	1,210	5.43	07+12	**70.9**
SSD+MobileNet	1,150	5.77	07+12+COCO	72.7
Pelee (ours)	1,210	5.43	07+12+COCO	**76.4**

3.3.3 Speed on Real Devices

We then evaluate the actual inference speed of Pelee on real devices. The speed on intel i7 is evaluated by the Caffe time tool. The speed on iPhone 6s and iPhone 8 are calculated by the average time of 100 images processed by the CoreML model. This time includes the image pre-processing time, but it does not include the time of the post-processing part (decoding the bounding-boxes and performing non-maximum suppression). Usually, post-processing is done on the CPU, which can be executed asynchronously with the other parts that are executed on mobile GPU. Hence, the actual speed should be very close to our test result. As can be seen from Table 7, the speed of Pelee is much faster than TinyYOLOv2 on iPhone 6s and Intel i7, but slower than TinyYOLOv2 on iPhone 8. Pelee runs at a 2.6 times faster speed on iPhone 6s than on Intel i7-6700K.

3.4 Results on COCO

We further validate Pelee on the COCO dataset. The models are trained on the COCO train+val dataset excluding 5000 minival images and evaluated on the test-dev2015 set. The batch size is set to 128. We first train the model with the learning rate of 10^{-2} for 70k iterations, and then continue

Table 7: Speed on Real Devices

Model	**mAP on VOC 2007 (%)**	**Speed** (FPS)		
		iPhone 6s	**iPhone 8**	**Intel i7-6700K**
YOLOv2-288	69.0	-	-	1.0
Tiny-YOLOv2	57.1	9.3	**23.8**	2.4
SSD+MobileNet	68	16.1	22.8	6.1
Pelee (ours)	**70.9**	**17.1**	23.6	**6.7**

training for 10k iterations with 10^{-3} and 20k iterations with 10^{-4}. Table 8 shows the results on test-dev2015. Pelee is not only more accurate than SSD+MobileNet (Huang et al. (2016b)), but also more accurate than YOLOv2 (Redmon & Farhadi (2016)) in both mAP@[0.5:0.95] and mAP@0.75. Meanwhile, Pelee is 13.6 times lower in computational cost and 11.3 times smaller in model size than YOLOv2.

Table 8: Results on COCO test-dev2015

Model	**Resolution**	**Computational Cost (MACs)**	**Model Size (Parameters)**	**Avg. Precision (%), IoU:**		
				0.5:0.95	**0.5**	**0.75**
SSD	300x300	34,360 M	34.30 M	25.1	43.1	25.8
YOLOv2	416x416	17,500 M	67.43 M	21.6	44.0	19.2
SSD+MobileNet	300x300	1,200 M	6.80 M	18.8	-	-
Pelee (ours)	304x304	1,290 M	5.98 M	**22.4**	**38.3**	**22.9**

4 Conclusion

Depthwise separable convolution is not the only way to build an efficient model. Instead of using depthwise separable convolution, our proposed PeleeNet and Pelee are built with conventional convolution and have achieved compelling results on ILSVRC 2012, VOC 2007 and COCO.

By combining efficient architecture design with mobile GPU and hardware-specified optimized runtime libraries, we are able to perform real-time prediction for image classification and object detection tasks on mobile devices. For example, Pelee, our proposed object detection system, can run 17.1 FPS on iPhone 6s and 23.6 FPS on iPhone 8 with high accuracy.

References

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2): 303–338, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016a.

Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678. ACM, 2014.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, pp. 1, 2011.

Kyoungmin Lee, Jaeseok Choi, Jisoo Jeong, and Nojun Kwak. Residual features and unified prediction network for single stage detection. *arXiv preprint arXiv:1707.05031*, 2017.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.

Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 3, pp. 7, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pp. 4278–4284, 2017.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.