GARBAGE-NET:GARBAGE CLASSIFICATION IN REAL-WORLD SET

REPORT FOR MACHINE LEARNING COURSE, SPRING 2020

Cathy **S. Y.**
cathyye0809@gmail.com

June 7, 2020

1 Introduction

Recently, garbage classification is becoming a trend in more and more cities. However, for the time being, people need to classify garbage manually, which is inconvenient and time consuming. Also, people lack of background knowledge may be confused about which kind of garbage a typical stuff is. So, people are in great need of an accurate automatic garbage classification system.

In this project, we make an attempt to tackle this problem with neural networks. We trained several models with different backbones(InceptionV3[1], VGG16[2], MobileNet[3]), and compared their performance. Without any bells and whistles, VGG16 with a Dropout layer reaches 100% accuracy on the test set. Also, we make an attempt to make our model work in real-world scenes. Our code and models are available at `https://github.com/YoruCathy/GarbageNet`.

We summarize our work as the following:

- We trained several models with different backbones(InceptionV3, VGG16, MobileNet), and compared their performance.
- We try different training skills, i.e, different batch size, input resolution, and learning rate schedule.
- We test the speed of the models and adopt MobileNet as the most suitable one for the Garbage classification task.
- We deploy it on Android to classify garbage in real-world.

2 Dataset Preparation

As is given in the project requirements, our dataset consists of six kinds of garbage, namely, cardboard, glass, metal, paper, plastic, and trash. Typical examples are shown in Fig.1. There are totally 2524 images in this dataset. We use 10% as testing set, and the rest 90% as training set.
Besides the given dataset, we also build our own dataset, named 'unseen' dataset, which is used for testing. It contains 41 images. Typical examples are shown in Fig.2. We try to make the objects from the same category in the two dataset different, so that we can test the generalize ability of the model.

Figure 1: Examples from the Given Dataset

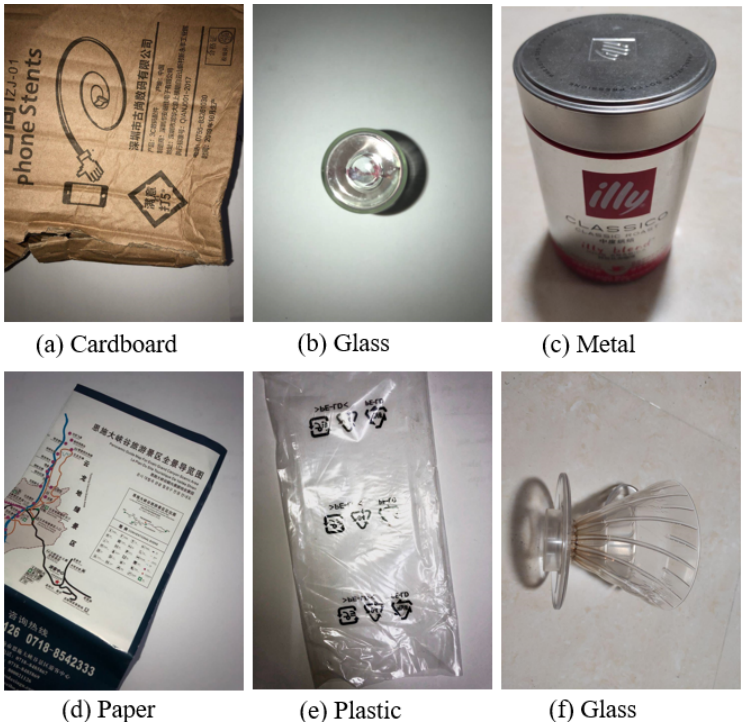


Figure 2: Examples from the Unseen Dataset

3 Network Structure

We build a simple network to address this garbage classification task, as is shown in Fig.3. The input image is resized into a certain size, for instance, 256$times$256. After the image is resized, it is fed into the CNN backbone. Here we try three backbones, namely InceptionV3[1], VGG16[2], and MobileNet[3]. Then the output is sent to a dropout layer. Finally, after a dense layer with activation function ReLU, and another one with softmax, the network gives a vector whose length is 6. It indicates the possibility of the input belonging to each class. The training details will be shown in Sec.4. In this section, we will roughly introduce the backbones and loss functions we use.

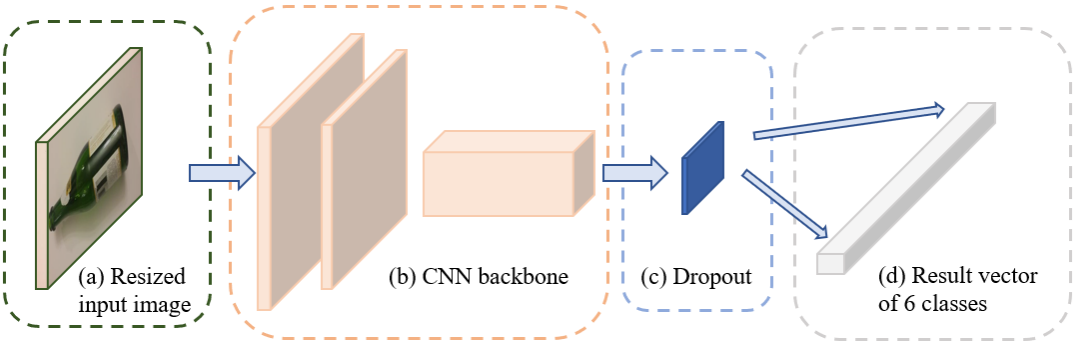


Figure 3: Network Structure

3.1 InceptionV3

The Inception network family is a classic series of networks that won a bunch of prizes. The inception module makes them good at classification, however, at the same time, makes them large. The structure of inception v3 is shown in Fig 4.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 4: Network Structure

3.2 MobileNet

MobileNet[3] is a network structure presented for mobile and embedded vision systems. It features depth-wise separable convolutions, which can effectively reduce computational cost and parameters. The MobileNet has a structure shown in Fig.5. It features short inference time and small model size.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 5: Network Structure

3.3 VGG16

VGG[2] is proposed to shown that increasing the depth can make the network perform better. It is also a classic network backbone which has been widely used. VGG16 has 16 hidden layers, as is shown in the D column in Fig. 6.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 6: Network Structure

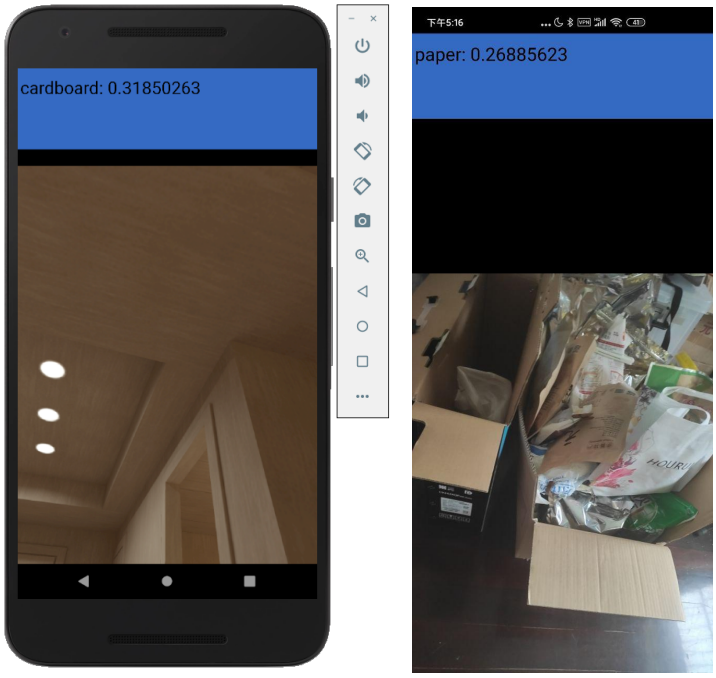
3.4 Loss:Categorical Cross Entropy

Categorical cross entropy is widely used in multi-class classification tasks. It is often used in company with softmax. We use categorical cross entropy as our loss function, as is defined by Eqn. 1, where p are the predictions, t are the targets, i denotes the data point and j denotes the class.

$$L_i = -\sum_j t_{i,j} \log\left(p_{i,j}\right) \tag{1}$$

4 Implement Details

We use keras[4], which is a high level API of tensorflow[5], to implement this network. Our code runs perfectly with Python 3.6.10, keras-gpu 2.3.1, and tensorflow-gpu 2.2.0. Pretrained weights on ImageNet[6] are used. We use random flip, shear, zoom, and rotate to augment our data.
As is shown in our code, a python class file named GarbageClassification.py is used as the core of our code. Example to train the model and infer with the model is provided in train.py and infer.py respectively. Also, we provide trained model with different backbones. The function of each function is introduced in the code comments, and can be also easily inferred from the function names. For the details, please refer to our codes, or read the documents on our GitHub.
For a garbage classification network, it is important to deploy it in real-word scenes, for example, an embedded system with a camera, or a mobile phone. We modify the Android demo application by Tensorflow, making it a garbage classification application of our GarbageNet, with MobileNet model. The interface is shown in 7. The application is included in our files. It can run on devices with API>=28, Android>=9.



(a) On virtual device (b) On real mobile phone

Figure 7: Android application interface

5 Experiment Result

5.1 Qualitative results and typical failures

We show the correctly classified examples in our **unseen dataset** in Fig. 8, and the mis-classified examples in Fig. 9. Here we use the VGG model with 100% accuracy to do the inference.

Figure 8: Examples from the Qualitative Results

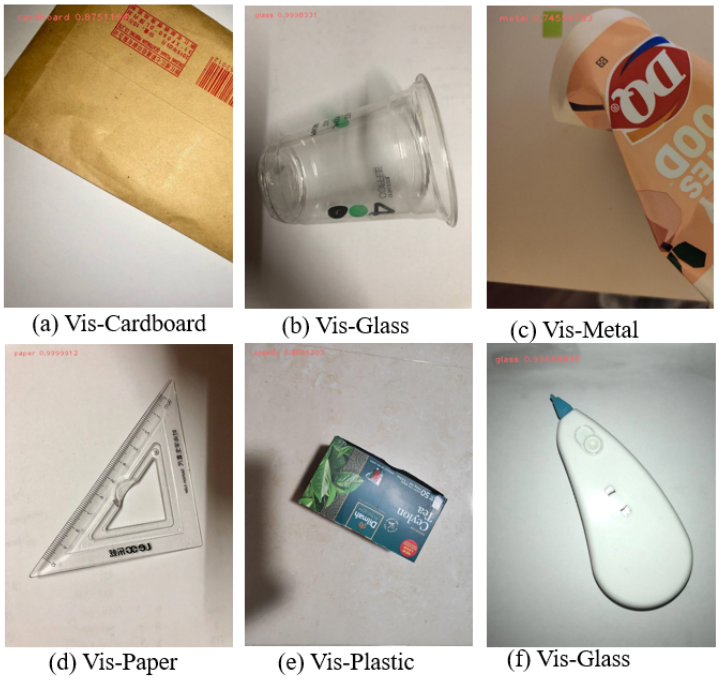


Figure 9: Examples from the Typical Failures

5.2 Speed

The inferring speed is tested on a GTX 1080 Ti GPU, with an Intel(R) Core(TM) i7-8700K CPU. We show our results in Table 5.2. As is suggested by [7], conventionally, when measuring the running speed of a inferring system, FPS and FLOPs are two common metrics. Since FPS is platform dependent, while FLOPs may or may not be linear with respect to actual running time due to a number of issues such as caching, I/O, hardware optimization etc, we just use their relative value for reference. The results suggest that MobileNet has the smallest FLOPs, and the shortest inference time. If it can perform similarly to the other models, it can be the most suitable one for real-world application.

Table 1: speed

Backbone	Inference Time (FPS)	FLOPs
Inception V3	2531	23.92M
Moblie Net	8263	4.24M
VGG16	7386	15.24M

5.3 Ablation Study

5.3.1 Different Backbone for Different Epochs

We train the network with the three backbones for 1k, 2k and 3k epochs. The result is shown in Table 5.3.1. It suggests that VGG16 is the best in accuracy, while MobileNet has a similar performance. So it is reasonable to implement MobileNet on Android Devices.

Table 2: Backbone

Backbone	Epoch	Accuracy
Inception V3	1k	0.89
	2k	0.94
	3k	0.94
Moblie Net	1k	0.92
	2k	0.93
	3k	0.94
VGG16	**1k**	**1.00**
	2k	1.00
	3k	1.00

5.3.2 ReLU vs ELU

ELU(Exponential Linear Unit) and ReLU(Rectified Linear Unit) are both activation functions of ANNs(Artificial Neural Networks). ELU is a combination of sigmoid and ReLU. Compared with ReLU, ELU still has outputs if input variable is negative, which adds to the robustness of networks. Study[8] shows that ELU performs better than ReLU in accuracy when classifying iamges, however, our experiments show a different result that ReLU is the better one in Table 5.3.2.

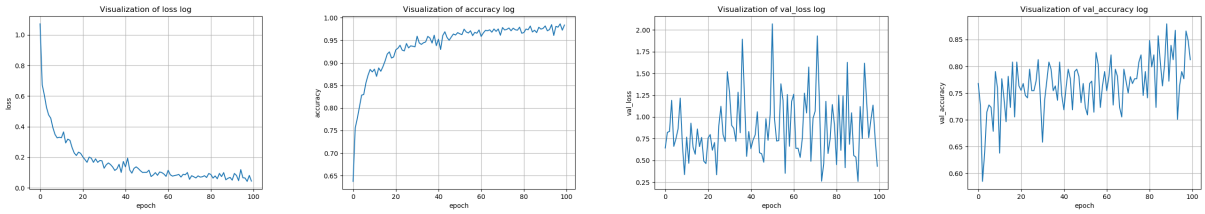
Table 3: ReLU and ELU as activation function of the Dense layer(trained for 100 epochs, with input size 224×224)

Backbone	ReLU	ELU
Inception V3	0.82	0.73
Moblie Net	0.88	0.79
VGG16	0.95	0.89

5.3.3 Learning Rate Schedule

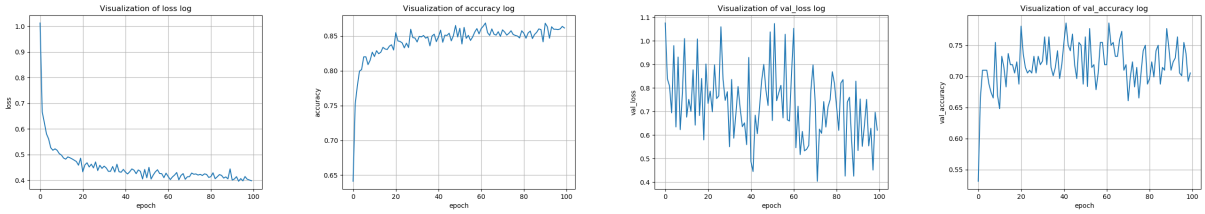
We try two kinds of learning rate schedules, namely exponential decay, and cosine learning rate. The log of train loss, train accuracy, val loss, and val accuracy on MobileNet are visualized. The training result of three backbones are shown

in Table 4. It suggests that exponential learning rate decay may not help, while cosine learning rate may be a little useful to improve the performance.



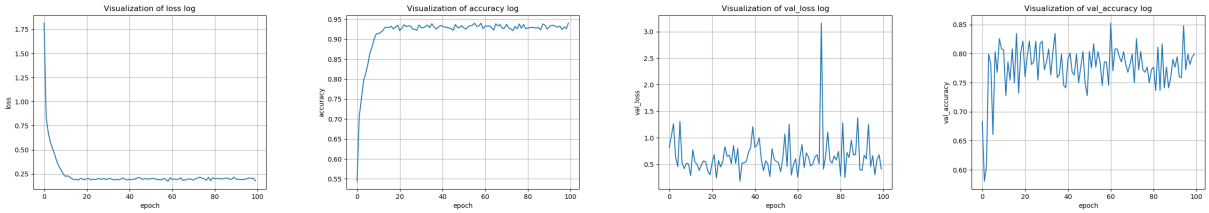
(a) Train Loss (b) Train Accuracy (c) Val Loss (d) Val Accuracy

Figure 10: Log of MobileNet, 100 epochs with normal LR schedule.



(a) Train Loss (b) Train Accuracy (c) Val Loss (d) Val Accuracy

Figure 11: Log of MobileNet, 100 epochs with LR Exponential decay of 0.1 at 4500 iteration.



(a) Train Loss (b) Train Accuracy (c) Val Loss (d) Val Accuracy

Figure 12: Log of MobileNet, 100 epochs with cosine learning rate schedule.

Table 4: Learning Rate Schedule(trained for 100 epochs, with input size 224×224)

Backbone	Original	Exponential decay	Cosine Learning Rate	Accuracy
Inception V3	✓			0.82
		✓		0.79
			✓	0.86
Moblie Net	✓			0.88
		✓		0.77
			✓	0.85
VGG16	✓			0.95
		✓		0.94
			✓	0.95

5.3.4 Input Size

We train the network with three kinds of input sizes of the images for 128×128, 192×192, and 224×224, in consistent with the ImageNet pretrained model. We show our results in Table 5.3.4. Obviously, larger input size can lead to better performance.

Table 5: Input Size (trained for 1k epoches, with batch size 32)

Backbone	224	192	128	Accuracy
Inception V3	✓			0.89
		✓		0.83
			✓	0.81
Moblie Net	✓			0.92
		✓		0.90
			✓	0.81
VGG16	✓			1.00
		✓		0.92
			✓	0.93

5.3.5 Batch Size

We train the network with four kinds of batch sizes for 8, 16, 32. We show our results in Table 5.3.5. Obviously, larger batch size can lead to better performance.

Table 6: Batch Size (trained for 1k epoches, with input size 224×224)

Batch Size	32	16	8	Accuracy
Inception V3	✓			0.89
		✓		0.89
			✓	0.87
Moblie Net	✓			0.92
		✓		0.90
			✓	0.88
VGG16	✓			1.00
		✓		0.99
			✓	0.97

6 Discussion: Generalization Ability

As is shown in Sec.5.1, the VGG16 model has 100% accuracy on the test set of the given dataset, however, it misclassifies many figs on the unseen dataset. Also, as in shown in Fig.7, on our Android Application, the model can do real-time inference, however, at a relatively low confidence.
We suppose the reason may be that:

- the given dataset is relatively small. 2k images are far from enough for real-world application.
- objects in the same category from the given dataset are similar. The lack of diversity makes the model not strong enough to handle real world cases.
- the fact that some information dimention is missed in images should not be ignored. For example, a metal box with colorful paints on it loos like plastic.

7 Conclusion

We try InceptionV3, VGG16 and MobileNet to classify garbage in this project. The MobileNet is the fastest one with a satisfying performance to be implemented on mobile devices. The VGG16 can reach a 100% accuracy on the given dataset.

We deploy our model on Android, making it able to do inference in real-world scenes. However, maybe due to lack of data, and lack of data diversity, the classification accuracy is not satisfying.
In conclusion, we think MobileNet may be able to classify garbage in real-world, if large amount of data is provided, and it is well trained. However, maybe some sensors, for example, conductivity measuring instrument can be adopted to help pure computer-vision-oriented methods to perform better.

References

[1] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[3] 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. *CoRR*, abs/1704.04861, 2017.

[4] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[5] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[6] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[7] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara Balan, Alireza Fathi, Ian S. Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2017.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.