Sketch-A-XNORNet: Binarization of Convolutional Neural Networks For Sketch Classification

Ayush Saraf
Paul G. Allen School of Computer Science & Engineering
ayush29f@cs.washington.edu

Abstract

We propose an extension to the deep convolution neural network called Sketch-A-Net, for free hand sketch classification, using weight binarization approximation techniques suggested in XNOR-Net. This would allow us to run the network on mobile devices with lower memory and compute power. In this paper we will attempt to quantitatively analyze the trade-offs between the classification accuracy and computational efficiency caused by these approximations. We will show that this will account for $\sim 32\times$ memory savings and $\sim 2\times$ faster convolutional operations on Sketch-A-Net. We will also show that the size of the pre-trained model (weights) is reduced by $\sim 18\times$.

1 Introduction

Free hand drawings and sketches have been one of the most effective forms of communication among humans since over 40,000 years in various forms including: cave paintings, art, line drawings etc. The more recent developments in more advanced interface technologies like touch-screen or stylus (input), and virtual or augmented reality (output) motivates sketching as an interface to communicate with the computers. The first step for us to be able to use it as a form of communication would be to build a real-time free hand sketch classification system that allows the computer to understand what subject matter the human is trying to communicate about.

Recent developments in deep learning and convolutional neural networks [1] have allowed the development for much more robust and high accuracy image classification models. The AlexNet [1] CNN which is a state of the art image classification model trained on ImageNet [2] provides great classification accuracies for photos, however it performs poorly on free hand sketches. Sketch-A-Net, also a deep CNN [3], proposes a different 8 layer network architecture specifically tailored towards free hand sketches and is trained on the largest know sketch data-set [4] published by TU-Berlin. The data-set has 20,000 labeled sketches with 250 classes with 80 examples for each class. This network claims to have an accuracy of 74.9% on a full scale Sketch-A-Net ensemble model (We would be using a much more simpler version of it in this paper).

The bigger challenge in the context human computer interaction is lack of high performance hardware resources like memory and GPUs in most devices. In order to provide a rich real-time interaction interface on mobile hand held or head mounted devices is that they require much more space and speed efficient implementations of these convolution neural networks. In order to achieve this, we will extend the Sketch-A-Net network to a Binary Weight CNN [5]. The Binary Weight network approximates convolutional operations using primarily addition and subtraction operations, which results in $\sim 2\times$ faster convolutions operation and $\sim 32\times$ memory savings. This allows for possibility of running these, otherwise computationally expensive networks, on low power devices like smart-phones, head mounted AR devices, and wearable devices.

Index	Layer	Type	Filter Size	Filter Num	Stride	Pad	Output Size
0		Input	-	-	-	-	225×225
1	L1	Conv	15×15	64	3	0	71×71
2		ReLU	-	-	-	-	71×71
3		Maxpool	3×3	-	2	0	35×35
4	L2	Conv	5×5	128	1	0	31×31
5		ReLU	-	-	-	-	31×31
6		Maxpool	3×3	-	2	0	15×15
7	L3	Conv	3×3	256	1	1	15×15
8		ReLU	-	-	-	-	15×15
9	L4	Conv	3×3	256	1	1	15×15
10		ReLU	-	-	-	-	15×15
11	L5	Conv	3×3	256	1	1	15×15
12		ReLU	-	-	-	-	15×15
13		Maxpool	3×3	-	2	0	7×7
14	L6	Conv(=FC)	7×7	512	1	0	1×1
15		ReLU	-	-	-	-	1×1
16		Dropout (0.50)	-	-	-	-	1×1
17	L7	Conv(=FC)	1×1	512	1	0	1×1
18		ReLU	-	-	-	-	1×1
19		Dropout (0.50)	-	-	-	-	1×1
20	L8	Conv(=FC)	1×1	250	1	0	1×1

Table 1: The architecture of Sketch-a-Net[3].

2 Methodology

In this section we will talk in detail about the key elements of our study. In the first sub-section we will talk about the Sketch-A-Net[3] architecture and what elements we decided to use to test our hypothesis about Binary Weight Sketch-A-Net. In the second sub-section we will talk about the how we binarize the network as suggested by Rastegari et al. in XNOR-Net[5]. Finally, in the last sub-section, we will explain how we implemented the Binary Weight Sketch-A-Net architecture in TensorFlow [6].

2.1 CNN Architecture: Sketch-A-Net

Designing CNN architecture is still an open question. However, there are certain rules of thumb that have been popularized in the recent years especially by Krizhevsky et al. in a AlexNet [1]. Most convolutional neural network architecture are composed of blocks of Convolutional-ReLU-MaxPool layers followed by fully connected layers and a classifier like softmax. AlexNet [1] was designed to perform image classification of the famous ImageNet [2] dataset, which consists of over 15 million labeled high-resolution images in over 22,000 categories.

Although AlexNet [1] performs classification very well on natural photo images, it doesn't perform as well on free-hand drawn sketches. Therefore, we choose to adapt the network architecture proposed by Yu et al. in Sketch-A-Net [3] (Table.1). Sketch-A-Net [3] architecture is inspired by AlexNet [1], and here is a summary of some of the important changes. A more detailed explanation could be found in [3].

- Larger First Layer Filters: The size of the filter is the most sensitive parameter for convolutional networks, because all other subsequent layers depend on it and it directly interacts with the original input. Unlike natural photo images, sketches lack texture information and the input images is very sparse. Therefore, having a larger filer size allows neurons in the first layer to capture more information about the input. Sketch-A-Net[3] uses a 15x15 filter size for the input images.
- No Local Response Normalisation: LRN [1] are usually used to deal with brightness and lighting in natural photo images, which is not required in the case of sketches.
- Higher Dropout: Dropout [7] is a form of regularization which prevents neural networks from over fitting. Since we are using a much smaller dataset compared to ImageNet [2] we use a higher dropout rate of 50%.

Since the goal of this study is to validate Binary Weight Sketch-A-Net, instead of using a multi-scale, multi-channel network ensemble as suggested in Sketch-A-Net [3], we simply use a simple multi-channel network, which we talk in detail about in 2.2

Figure 1: Illustration of stroke ordering for a sketch from Harp class.

2.2 Stroke Order Modelling

A free hand sketch, unlike natural photo images, comprises of a series of strokes which can provide much more information about the sketch. Even though the most common sketch datasets like the one we use in this study from TU-Berlin [4] contains this information in the form of SVG (Support Vector Graphic) images, not many studies utilize it. Modeling stroke order as part of the input to the learning algorithm is in fact very practical as in most real world scenarios it is very easy to gather this information during inference time from mobile devices like smart phones or even augmented reality head mounted displays.

The Sketch-A-Net [3] proposes a multi-channel stroke order modelling technique that allows the convolutional neural network to take advantage of this information. The proposed model discretize the original image into 3 different images. Further, it forms 2 more images from a combination of two images from the 3 dicretized one. Finally, we stack the 5 images along with the original image to form a 6 channel image. Therefore, the deep CNN Sketch-A-Net takes an input with 6 channels. The Fig.1 shows an example of the stroke order modelling for a sketch from the harp class. This allows the network to automatically learn stroke ordering from back propagation.

A more recent update from the Sketch-A-Net [8] performs a more sophisticated single-channel stroke order modelling[1]. It is designed after the assumption that when drawing sketches, the longer outlines are usually drawn first and then the details with shorter stoke length. Which allows us to model a probability function to make less detailed sketches from a more detailed sketch by having different thresholds on the following probability function for keeping the ith stroke in the resulting sketch.

$$Pr_i = \frac{e^{\alpha o_i}/e^{\beta l_i}}{Z}$$
$$Z = \sum_i e^{\alpha o_i}/e^{\beta l_i}$$

2.3 Binarization of Sketch-A-Net

In this section we will explain how we used the binarization techniques, suggested by Rastegari et al. [5], to train our Sketch-A-Net model. The biggest bottle neck of any convolutional neural network is the extremely expensive convolutional operation. In order to reduce the complexity of this convolutional operation XNOR-Net[5] presented 2 major approximation techniques: (i) Binary Weight Network and (ii) XNOR-Networks[2] (binary weights and inputs).

2.3.1 Binary Weight Network

This is the first approximation technique that binarize the weights of a network which allows convolutional operations with only addition operations instead of much more expensive multiplication operations. Therefore, the convolutional operation could be approximated[3] by the following operation:

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\alpha$$

[1]Although, it is a more sophisticated form of stroke ordering, we don't use it in our model because of unavailability of a pre-trained model, limited hardware resources and time.

[2]Not covered in this study because of limited time and compute resources

[3]The XNOR-Net [5] ignores the bias term, therefore we only binarize weights and keep the biases as 32-bit floats

In the above equation, we assume $\mathbf{W} \in \mathbf{R}^{c \times w \times h}$, $\mathbf{B} \in \{-1, +1\}^{c \times w \times h}$ and $\alpha \in \mathbf{R}^{+}$. In order to achieve the approximation, we perform the following optimization.

$$J(\mathbf{B}, \alpha) = ||\mathbf{W} - \alpha \mathbf{B}||^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$J(\mathbf{B}, \alpha) = \alpha^2 \mathbf{B}^T \mathbf{B} - 2\alpha \mathbf{W}^T \mathbf{B} + \mathbf{W}^T \mathbf{W}$$

Since $\mathbf{W} \in \mathbf{R}^{c \times w \times h}$, $\mathbf{B} \in \{-1, +1\}^{c \times w \times h}$, $\mathbf{B}^T \mathbf{B} = n$ and $\mathbf{W}^T \mathbf{W} = c$ are also constant. Therefore,

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha \mathbf{W}^T \mathbf{B} + c$$

Since $\alpha \in \mathbf{R}^{+}$, we can ignore it and the constant from the optimization.

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}} \, \mathbf{W}^T \mathbf{B} \;\; s.t. \;\; \mathbf{B} \in \{+1, -1\}^n$$

$$\mathbf{B}^* = sign(\mathbf{W})$$

$$\alpha^* = \frac{\mathbf{W}^T sign(\mathbf{W})}{n} = \frac{\sum_i |\mathbf{W}_i|}{n} = \frac{||\mathbf{W}||_{l1}}{n}$$

Therefore, the optimal solution for the binary weights can essentially be calculated simply by a $sign$ and a $l1$ norm operation. The two combined gives us the binarization operation for binarizing the weights. However, in order to perform the backward pass we will need to calculate a gradient for this operation. Since, the operation is not differentiable we use the gradient approximation as suggested in [9].

$$\frac{\partial sign}{\partial r} = r1_{|r| \leq 1}$$

$$\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial \tilde{W}_i}(\frac{1}{n} + \frac{\partial sign}{\partial W_i}\alpha)$$

The above equation gives us the gradient we will use for the back propagation. More details about the specifics of these assumptions could be found in [5] [9]. Finally, we use this binarization operation on all the weights for convolutional layer 2 to 7.

2.4 TensorFlow Implementation

In order to implement and train this deep convolutional neural network we choose to use TensorFlow's [6] Python API. It is a machine learning library that uses the concept of a computational graph in which nodes represent mathematical operation and the edges represent flow of tensors or multi-dimensional data arrays. There are multiple reasons why we choose to use TensorFlow for implementing this convolutional neural network.

- It allows us to easily represent the convolutional neural network in an abstract and flexible graph like structure. Having such flexibility helps us iterate on our training process much more quickly.
- It abstracts away the complexity of using system resources like GPUs and uses the computational graph to optimally distribute the processes among all the available system resources.
- It provides simple and efficient check-pointing API that allows us to pause and continue learning from any checkpoints without having to implement anything ourselves.
- It provides simple to use logging and visualization APIs which allow us to track our learning progress in real time. Fig.2 shows the graph visualization of our network constructed by TensorBoard.
- It is till date the only machine learning library designed and maintained by a major software company (Google), this gives it an advantage of having great developer support.

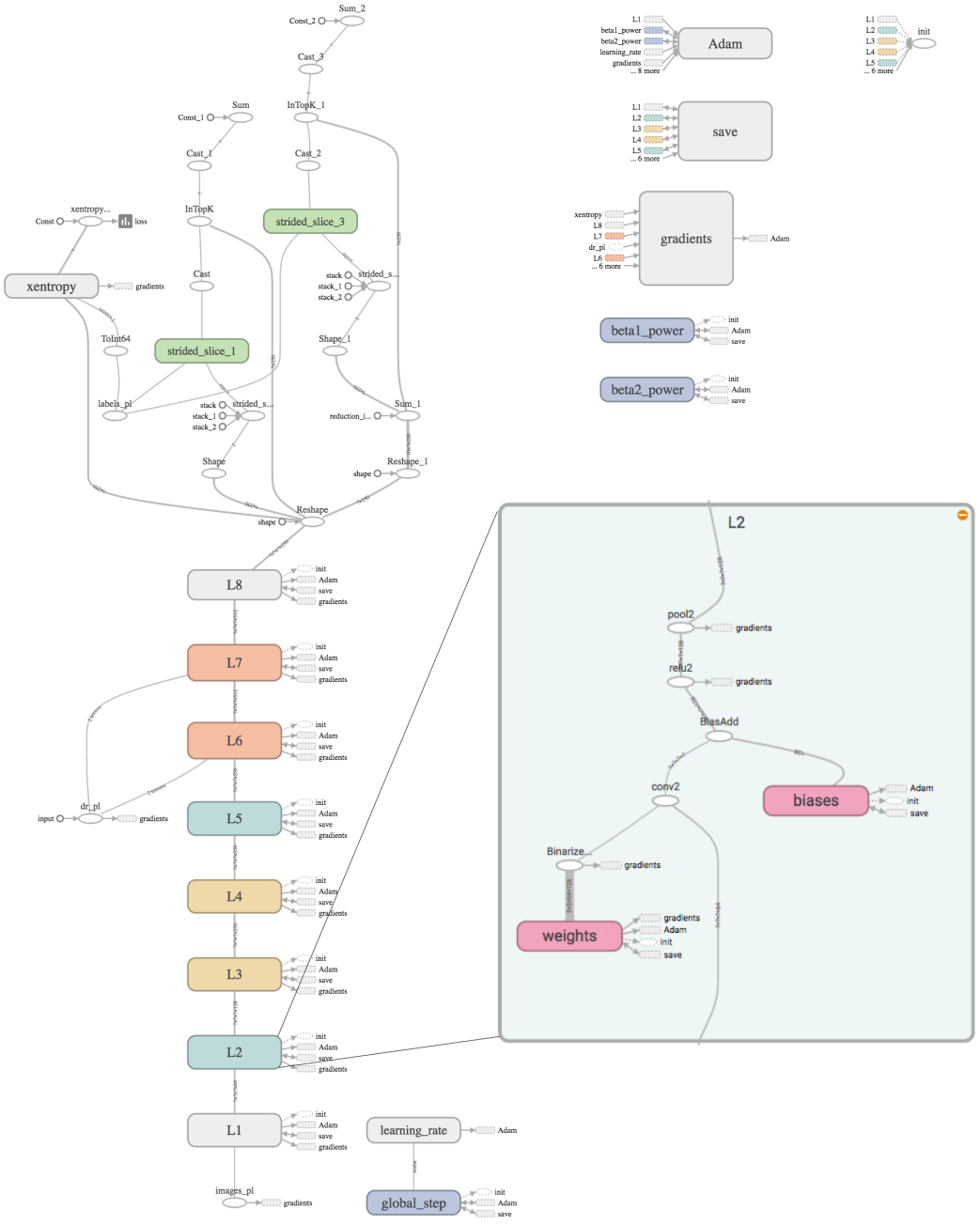

Figure 2: TensorFlow inference graph for Binary Weight Sketch-A-Net

2.4.1 Creating Binary Ops

Even though TensorFlow [6] is one of the most popular machine learning library, it does not yet support binary operations as suggested in XNOR-Net [5]. Although there have been efforts, the work in binary neural network is still not popular in real world applications, and therefore not inherently supported by most widely used machine learning libraries.

Therefore, in order to implement binary convolutional neural networks we will first have to implement new Ops in TensorFlow [6] that perform the binarization and convolutional operation that takes advantage of binarization and does convolutional without multiplication. It is possible to write your own Ops in TensorFlow [6] in both C++/CUDA or Python. In this study we choose to write our weight binarization Op in python using py_func, a function available in TensorFlow [6] API that allows us to write TensorFlow Ops in python. The following is the implementation of the weight binarization operation:

```
# forward pass
def f(x):
    alpha = np.abs(x).sum(0).sum(0).sum(0) / x[:,:,:,0].size
    y = np.sign(x)
    y[y == 0] = 1
    return y * alpha

# backward pass
def df(op, grad):
    x = op.inputs[0]
    n = tf.reduce_prod(tf.shape(x[:,:,:,0])[:3])
    alpha = tf.div(tf.reduce_sum(tf.abs(x), [0, 1, 2]),
        tf.cast(n, tf.float32))
    ds = tf.multiply(x,
        tf.cast(tf.less_equal(tf.abs(x), 1), tf.float32))
    return tf.multiply(grad,
        tf.add(tf.cast(1/n, tf.float32), tf.multiply(alpha, ds))))
```

It is important to note that this Op just allows us to validate the results of the binarization and not actually results in any performance improvement. In order increase the performance, we would have to implement the new convolutional operation that takes advantage of the binarized weights. This requires a lot of changes to the current version of TensorFlow and is out of the scope for this study.

3 Experiments

In this section, we will discuss the details and results of all the experiments we performed to analyze the trade offs between efficiency and accuracy of binarization of Sketch-A-Net. Firstly, we will discuss the data set and the data augmentation techniques used to train these networks. Secondly, we will talk about the training settings used to train the network. Thirdly, we will discuss trade-offs in accuracies of the networks. Finally, we will perform some visualization techniques to get a sense of what the network is really learning. The source code for all the experiments could be found on github[4]

3.1 Dataset & Data Augmentation

Dataset: We use the TU-Berlin sketch dataset [4] to train and evaluate our model. The dataset is one of largest and most widely used free hand sketch dataset. The dataset was collected using Amazon Mechanical Turk (AMT) from 1,350 participants and it consists of 250 categories with 80 sketches per category. The images are available in SVG (Support Vector Graphic) format which contains the stroke order information that we exploited in 2.2. We then re-size all the images to 256 x 256 pixel images. Further, we split the dataset into a training set of 13,500 images and a testing set with 6,500 images. All the categories have equal number of examples in both sets which accounts for any bias towards a particular class.

[4] https://github.com/ayush29feb/Sketch-A-XNORNet

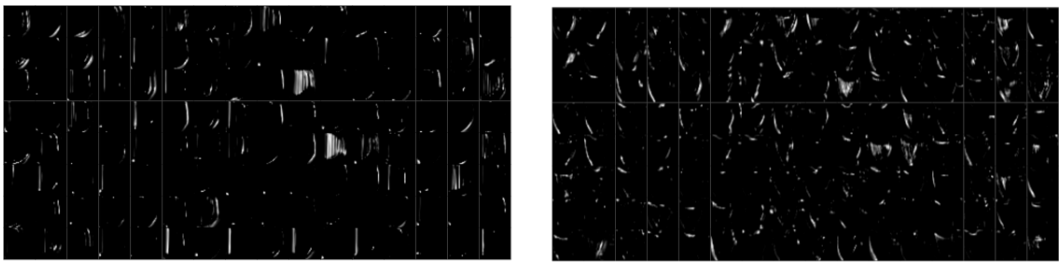

Figure 3: Representation of L2 ReLU activation maps in Binary Weight Sketch-A-Net (left) and Vanilla Sketch-A-Net (right)

Data Augmentation: In order to avoid over fitting, we use various data augmentation techniques by performing number of transformations. This adds randomness to the training dataset and results in much better learning with less over fitting. We perform the following transformation for each image: (i) We first take a random crop of 225 x 225 pixels image from the original 256 x 256 pixels image, (ii) we perform horizontal flips to images with a probability of 0.55 and (iii) we perform random rotations between [-5, 5] degrees with a probability of 0.55. This results in increasing training dataset by a factor of 32 * 32 * 2 * 11 = 22,528. Further, for the testing set we perform 5 unique crops and a horizontal flip for each of these crops which results in 10 unique images. We use all of these images and pick the category with the highest votes as the prediction.

3.2 Training

In order to reproduce Sketch-A-Net we used the pre-trained model, trained on dataset of 256 x 256 pixel images, provided by Yu et al. on their website[5] available in Matlab format. Further, in order to train the binarized networks we used the pre-trained model as initialization for weight and bias variables. We then use AdamOptimizer [10] with an exponentially decaying learning rate and default parameters. We initialize the learning rate with a value of 0.001, the a decay step size with 1000, and decay rate of 0.96. We set the batch size to be 54, which makes a single epoch 250 steps. The network usually converges around 40-60 epochs or 10,000-15,000 steps. We also used a NVIDIA GeForce GTX 1060 6GB GPU for training.

3.3 Results

Sketch-A-Net: After evaluating the pre-trained Sketch-A-Net model we achieved a top-1 accuracy of 62.02% and a top-5 accuracy of 90.08% on the test set. These accuracy are much lower compared to the full scale Sketch-A-Net multi-scale multi-channel Join Bayesian network ensemble which claims to result a top-1 accuracy of 74.9%.

Binary Weight Sketch-A-Net: After evaluating the resulting trained Binary Weight Sketch-A-Net model we achieved a top-1 accuracy of 58.11% and a top-5 accuracy of 82.35% on the test set. These accuracies compared to our results from Sketch-A-Net are fairly close given that we used the weight binarization approximation. This shows that even with such an extreme constraint the network was able to learn fairly well.

3.4 Visualizations: ReLU Activations

One of the biggest challenges for neural networks research has been that we don't completely understand what these neural networks are learning and on what criteria they are making their predictions. In order to get some intuition about whats happening behind the scenes, we attempt to visualize the intermediate states of the input before it gets classified. More specifically, we visualize the output of the early ReLU activation layers. Fig.3 shows the output of the 2nd ReLU activation layer for both the normal Sketch-A-Net and Binary Weight Sketch-A-Net. Although it is still very hard to gather much from these visualizations, we can see there are certain neurons (activation maps) which get

[5] `www.eecs.qmul.ac.uk/~tmh/downloads.html`

excited about the strings of the harp, it is much more clearly visible in the case of Binary Weight Sketch-A-Net. There are few other neurons which are detecting certain edges of the sketch and not the others.

4 Conclusion

We have proposed an extension to Sketch-A-Net [3] with weight binarization techniques suggested by Rastegari et al. in [5] which we refer to as "Binary Weight Sketch-A-Net". We validated our hypothesis of binarizing weights by testing it on a simpler version of Sketch-A-Net [3] which only loses 4 points on top-1 accuracy compared to a traditional real valued Sketch-A-Net [3] CNN. This allows our weights memory usage to reduce to 1.82MB compared to original 32.72MB which is over $\sim 18\times$ memory savings for a pre-trained model. Further, it shows that it will give $\sim 32\times$ memory savings and $\sim 2\times$ faster convolution operations during inference time. This accounts for better performance on mobile devices with low memory and compute power.

5 Future Work

Due to the limited time and compute resources we were only able to validate Binary Weight Sketch-A-Net model. However, there is still a lot of work required in order to finish this study. Some of the key next steps would be as follows:

- In this paper we only validated Binary Weight Sketch-A-Net. The next step would be to investigate into altering the Sketch-A-Net [3] such that it follows the requirements of XNOR-Net [5]. We attempted for a naive implementation of Sketch-A-XNORNet but didn't achieve any promising results.
- The most important piece to consider is it to implement the network with a convolutional operation that uses just addition or bit-counting. Without this new convolutional operation the system will not be taking full advantage of binarization in terms of performance.
- Finally, a last step would be to binarize the full scale network ensemble suggested in Sketch-A-Net [3] with better stroke ordering and deformation techniques as described in [8].

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] Qian Yu, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-net that beats humans. In Mark W. Jones Xianghua Xie and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 7.1–7.12. BMVA Press, September 2015.

[4] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[5] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals,

Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[8] Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M. Hospedales. Sketch-a-net: A deep neural network that beats humans. *International Journal of Computer Vision*, pages 1–15, 2016.

[9] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.