Sketch-A-XNORNet

Binarization of CNN for Sketch Classification

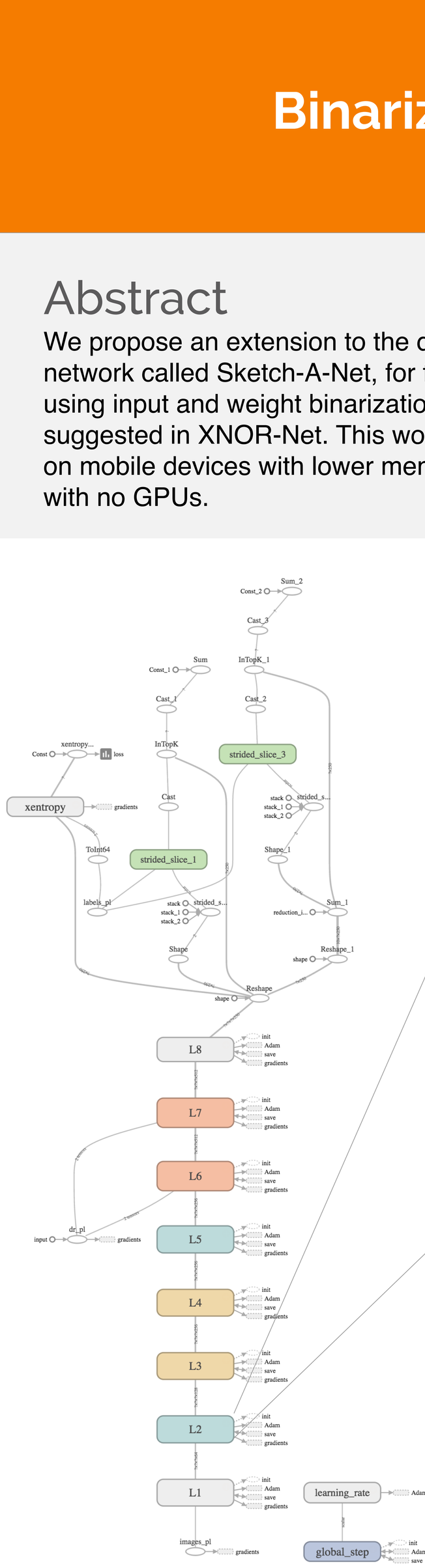
Ayush Saraf

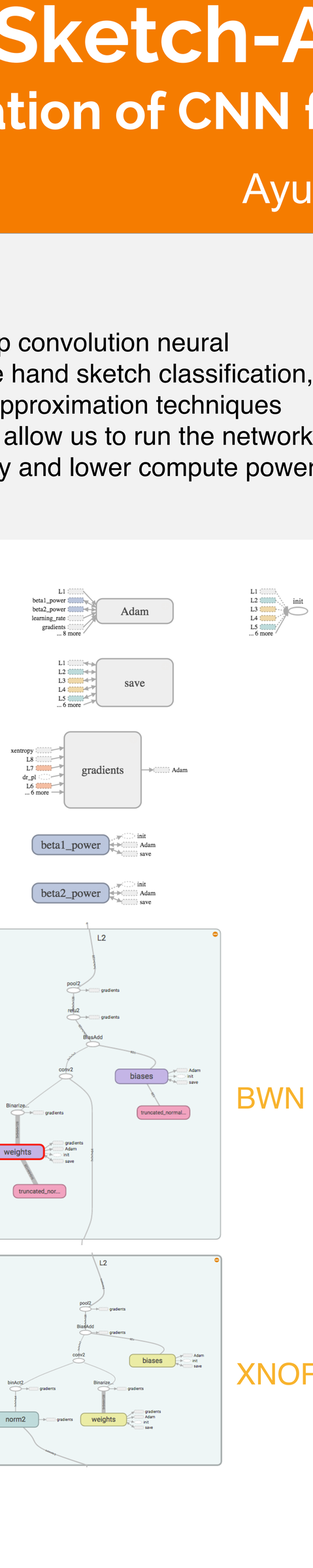
Abstract

We propose an extension to the deep convolution neural network called Sketch-A-Net, for free hand sketch classification, using input and weight binarization approximation techniques suggested in XNOR-Net. This would allow us to run the network on mobile devices with lower memory and lower compute power with no GPUs.

Approach

- Reporoduce the results of Sketch-A-Net in TensorFlow.
- Write a DataLayer for easy access to the dataset with augmentation
- Write Binarization Ops in TensorFlow
- Retrain the new BWN or XNOR networks using.
- Visualize ReLU activation layers for better understanding





BWN

XNOR

Binary Ops

$$\mathbf{B} \approx sign(\mathbf{W}) \frac{||\mathbf{W}||_{l1}}{n}$$

$$\mathbf{H} \approx sign(\mathbf{I})$$

```python
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

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)))
```

```python
def f(x):
    y = np.sign(x)
    y[y == 0] = 1
    return y

def df(op, grad):
    x = op.inputs[0]
    alpha = tf.cast(tf.less_equal(tf.abs(x), 1), tf.float32)
    return tf.multiply(grad, alpha) # grad * alpha
```

Training

Pretraining

We use the Sketch-A-Net weights for initialization to train the binary weights model

Starts with 10% accuracy

Optimizer

We use AdamOptimizer with a exponentially decaying learning rate

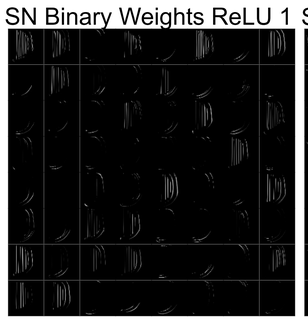
Initilize:
lr = 0.01
decay steps = 1000
initial global step = 10000

Hardware

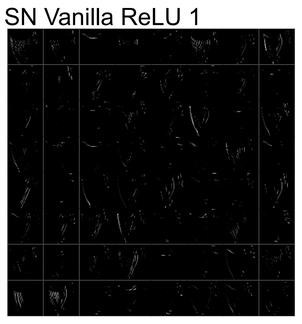
We used a NVIDIA GeForce GTX 1060 6GB GPU for training

Activation Layer Visualization

SN Binary Weights ReLU 1



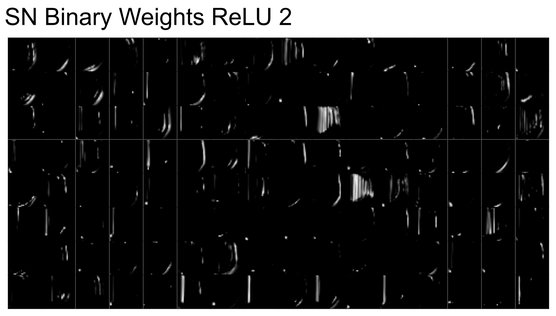
SN Vanilla ReLU 1



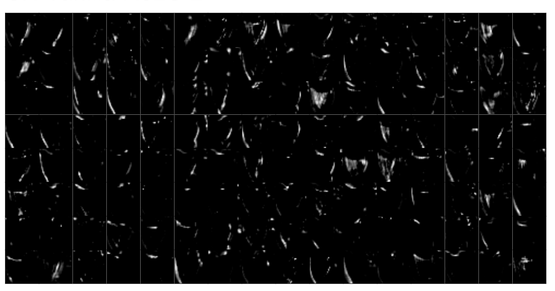
Index	Layer	Type	Filter Size	Filter Num	Stride	Pad	Output Size
0		Input	-	-	-	-	225×225
1	L1	Conv	15×15	64	3	0	71×71
2		ReLU	-	-	-	-	71×71
3		Maxpool	3×3	-	2	0	35×35
4	L2	Conv	5×5	128	1	0	31×31
5		ReLU	-	-	-	-	31×31
6		Maxpool	3×3	-	2	0	15×15
7	L3	Conv	3×3	256	1	1	15×15
8		ReLU	-	-	-	-	15×15
9	L4	Conv	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

Binarized Layers

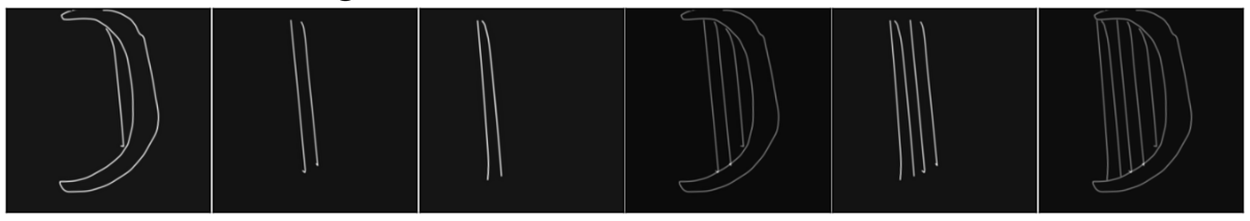
SN Binary Weights ReLU 2



SN Vanilla ReLU 2



Data & Data Augmentation



Temopral/Order Property

$$Pr_i = \frac{1}{Z} e^{\alpha * o_i} / e^{\beta * l_i}$$

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

Train Randomness

256x256 images
225x225 crops
random horizontal flips
random [-5,5] deg rotation

Testing

5 fixed crops x 2 flips
= 10 images/datapoint
Add scores from all =>
Predict the best score

Results

SN Binary Weights	SN Vanilla
82.35% Top-5	90.08% Top-5
58.11% Top-1	62.02% Top-1

Future Work

1. In order to better understand the difference between what the 2 networks are learning we would like to using other visualization techniques like deconvolution or opitmized images.
2. Further, due to limited time we only trianed a binary weight network. The work related to training a XNOR network is still incomplete.