JULIA IMAGE COLORIZATION USING KNET

Jeffrey Lu and Kevin Liu

6.338/18.337 Fall 2017

MOTIVATION

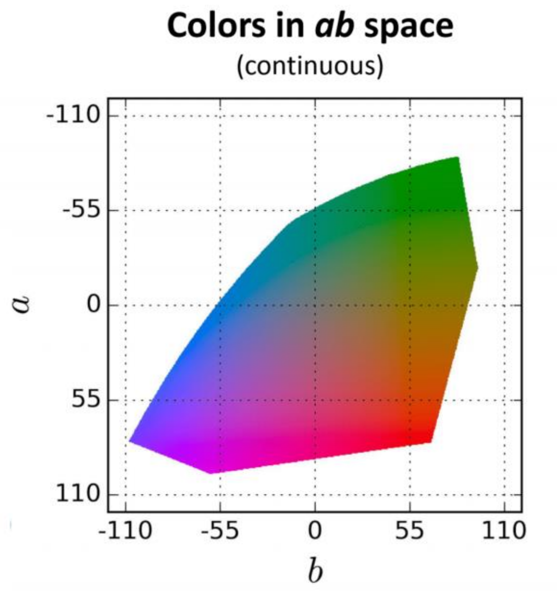
- Image colorization for 6.869 Computer Vision (Jeffrey)
- Cool application of deep learning
- Restoring old black and white photos
- Abstract experiment: no “right” answer
- **Experiment with deep learning in Knet and Julia**
- **Test ease of using Julia/Knet on AWS GPU**
- Based off *Zhang et. al.*’s image colorization paper

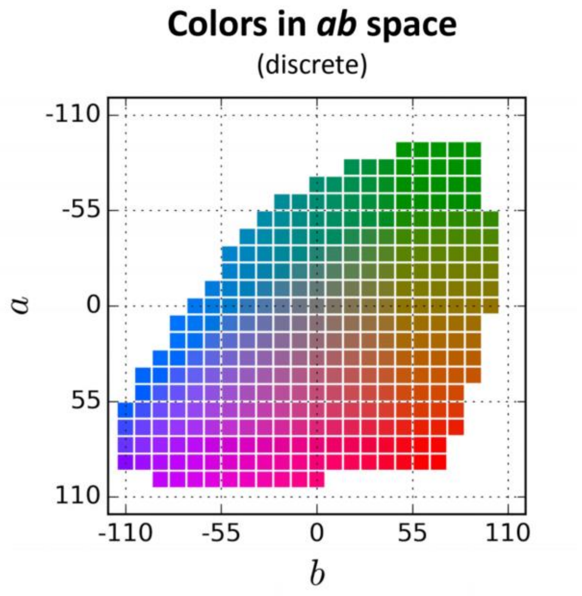
GOALS

- Given input black and white image
- Generate plausible color version
- Two possible methods
  - Supervised vs. unsupervised colorization
- Be able to colorize any input image of correct dimension

APPROACH

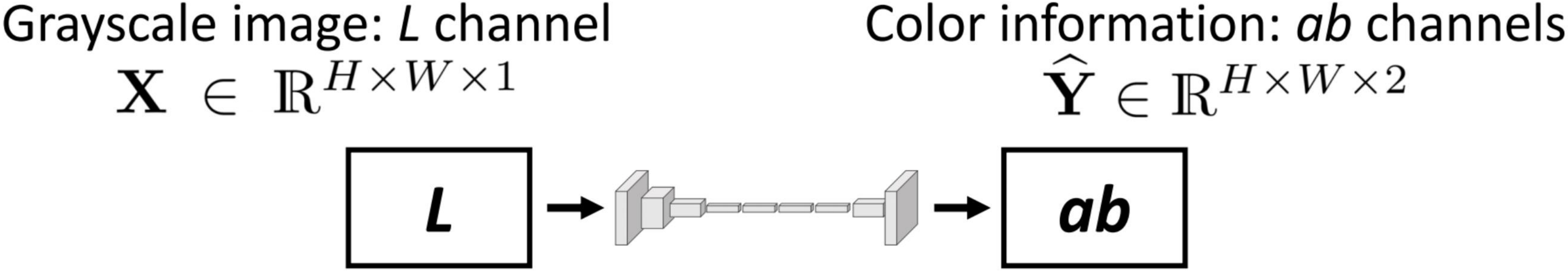
- Typically images represented in RGB
- Will use Lab color space
  - Only need to predict two values a and b, not 3
  - L channel gives lightness, same as grayscale value of input, no need to predict
  - Discretize Lab space to 18 by 18 buckets of (a,b) combinations
- Colors.jl to convert RGB to Lab





APPROACH (CONT.)

- Bucket each pixel in ground truth into ab bins
- Feed in L channel image as input to network
- Predict probability distribution of ab bins of each pixel
- Measure loss between ground truth and ab predictions
- Colorize image using highest probability ab bin for each pixel

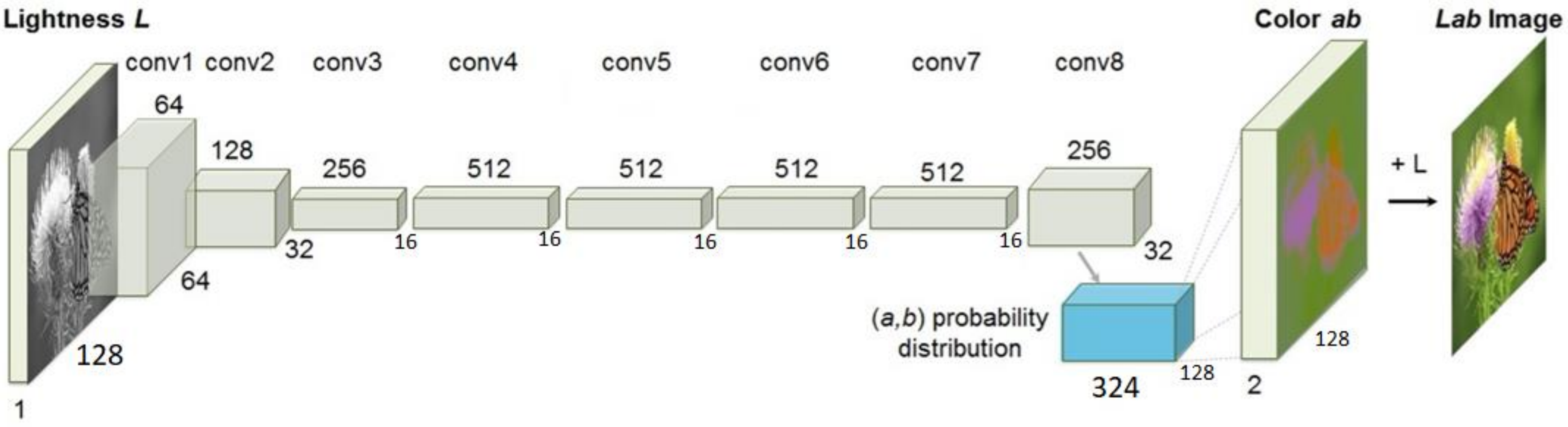


DATASET

- Miniplaces dataset
  - 100k images
  - Covers over 100 scenes
  - 128 by 128 images
  - Each image labelled with scene category
    - Can be used to improve network
- Subset of Places2 dataset from MIT CSAIL Computer Vision group

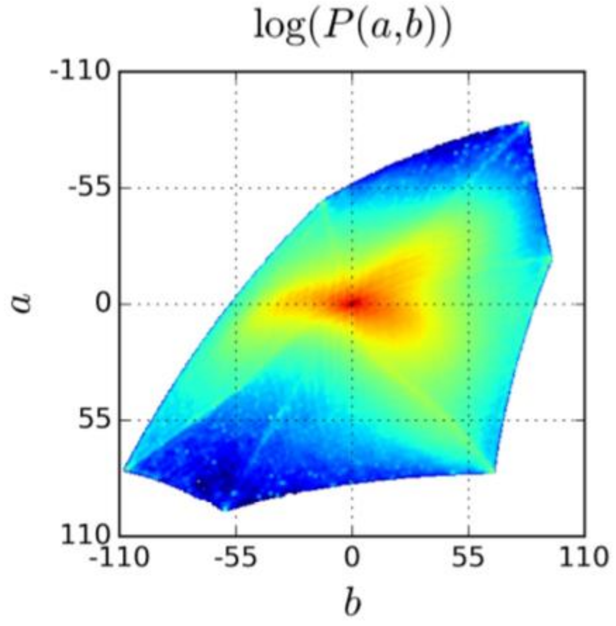
NETWORK ARCHITECTURE

- 22 convolutional layers separated into 8 groups
    - ReLU activation
    - Downsampling with stride 2 for dimension reduction
    - Upsampling at end to recover dimensions
- Randomly initialized parameters and biases



CLASS REBALANCING

- Distribution of ab buckets in images is very skewed
- If unadjusted, loss will be dominated by these buckets
- Loss of each pixel weighted by $-\log(p_{ai,bj})$
  - p is proportion of total pixels in training set that lie in ab bucket (i, j)



LOSS FUNCTION

- Single image loss

$$Loss(\hat{Y}, Y) = -\sum_{h,w} -\log(p_{ai,bj}) \cdot \hat{Y}_{h,w,ai,bj}$$

where (i, j) is true ab bin for pixel

- Loss of minibatch is sum of losses of images in batch
- Use loss to backpropagate and update weights of network

CHALLENGES

- Training on GPU instance
  - For speed, loss calculation needs to be vectorized
  - Problems with Knet and Autograd on GPU
- Size of training set – 100,000 images of size 128x128
  - Cannot fit in RAM
- Number of parameters in model
  - Long training time
- No visualization during training like Tensorboard