Cdiscount's Image Classification

MinWoo Byeon
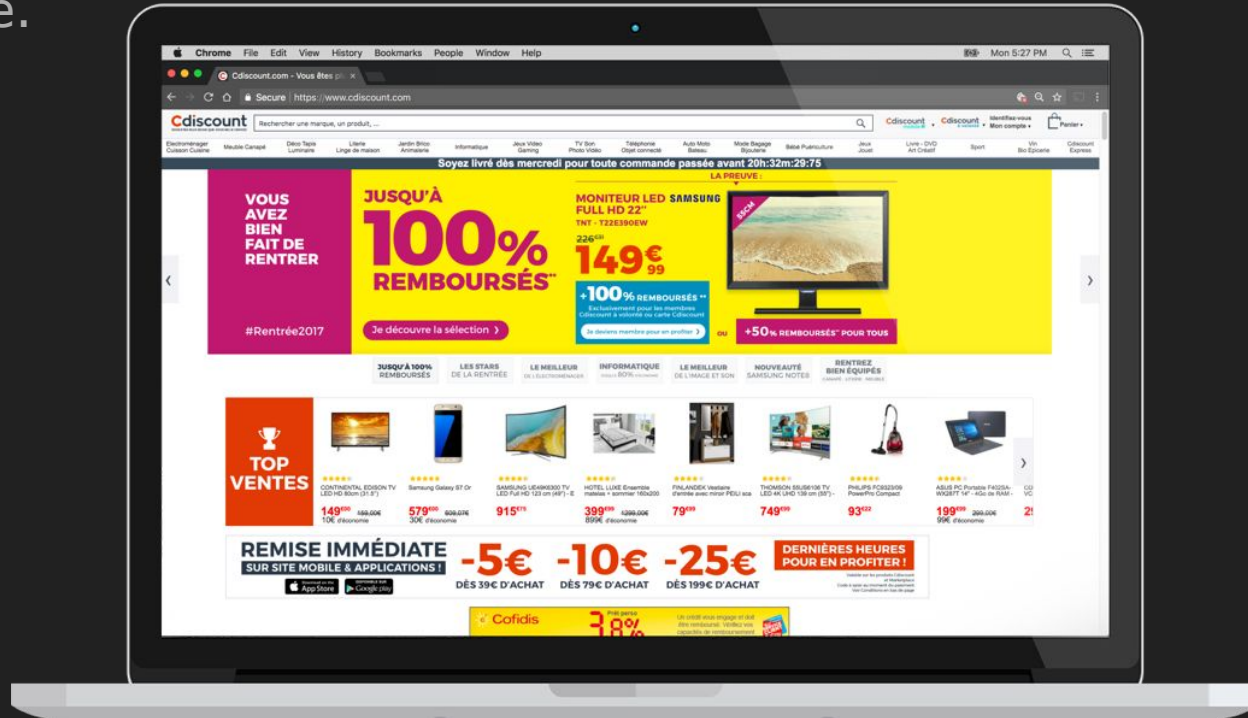
About me

- MinWoo Byeon
- Software Engineer at Kakao Corp
- Lives in Jeju Island, South Korea

- Bachelor's degree, Computer Engineering
- Interested in Machine Learning, Algorithms and Competitions

- Kaggle: https://www.kaggle.com/mwbyeon
- LinkedIn: https://www.linkedin.com/in/minwoo-byeon/

Cdiscount

- Cdiscount.com is the largest non-food e-commerce company in France.

Competition Overview

- https://www.kaggle.com/c/cdiscount-image-classification-challenge
  - In this challenge you will be building a model that automatically classifies the products based on their images. As a quick tour of Cdiscount.com's website can confirm, one product can have one or several images. The data set Cdiscount.com is making available is unique and characterized by superlative numbers in several ways:
    - Almost **9 million products**: half of the current catalogue
    - More than **15 million images** at **180x180** resolution
    - More than **5000 categories**: yes this is quite an extreme multi-class classification!

Competition Environments

- Python3 (with PyCharm)
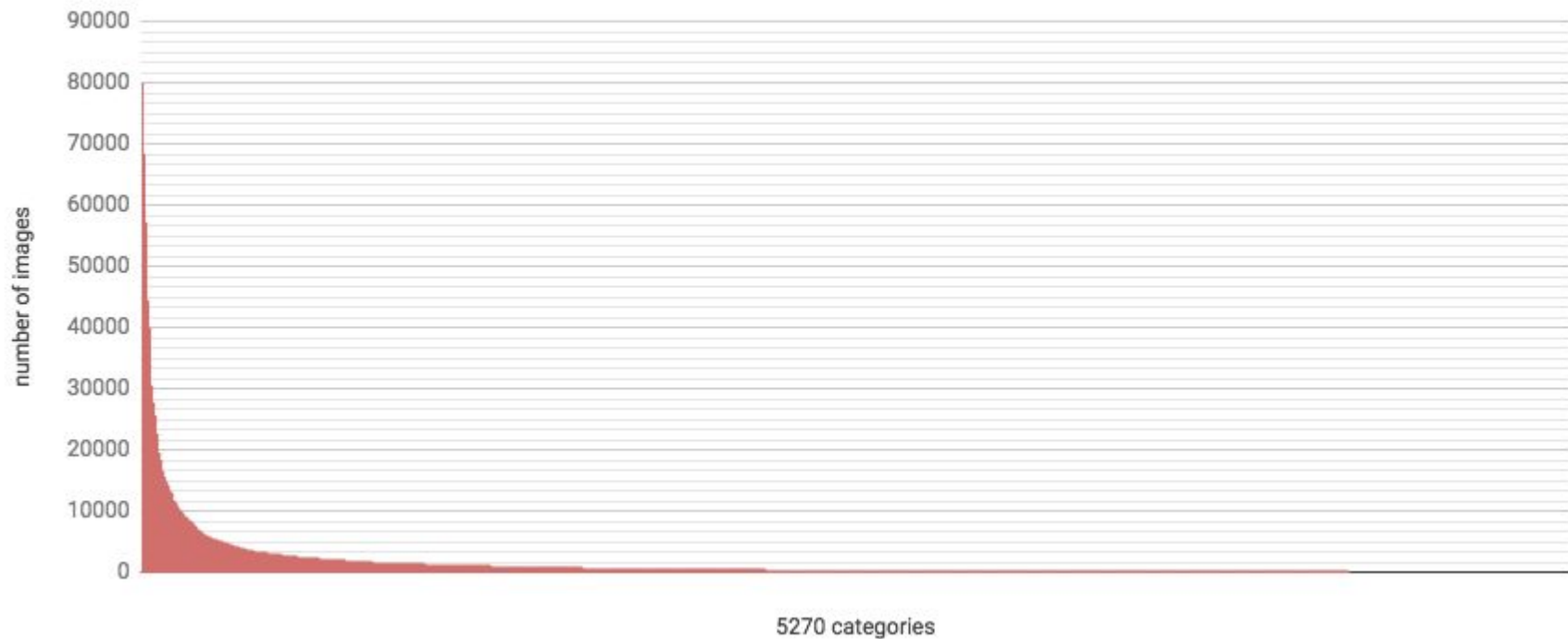- MXNet 0.12.0 (GPU enabled)
- Pascal based 8-GPU Machine

Dataset

Dataset

- Training Set
  - train.bson, 58.2 GB
  - 7,069,896 products
  - 12,371,293 images
    - each product contains between 1-4 images
- Test Set
  - test.bson, 14.5GB
  - 1,768,182 products

Imbalanced Dataset

Catetory Distribution

number of images

90000
80000
70000
60000
50000
40000
30000
20000
10000
0

5270 categories

Training and Validation

- Split the BSON file to Training and Validation
- Split products in the `train.bson` to `train_train.bson` and `train_valid.bson`
  - randomly selected with seed
  - Training(0.95) : Validation(0.05)

Create different datasets

- DATASET A
  - split products to 0.95(training) : 0.05(validation)
  - random seed: 12648430 `(0xC0FFEE)`
- DATASET B
  - split products to 0.95(training) : 0.05(validation)
  - random seed: 1
- DATASET C
  - split products to 0.95(training) : 0.05(validation)
  - random seed: 12648430 `(0xC0FFEE)`
 - remove duplicated images from DATASET A (it can reduce training time)

CNNs

- DenseNets
  - https://arxiv.org/abs/1608.06993
- Dual Path Networks
  - https://arxiv.org/abs/1707.01629
- ResNext
  - https://arxiv.org/abs/1611.05431
- SE-ResNext
  - https://arxiv.org/abs/1709.01507

DenseNets

- Paper
  - Densely Connected Convolutional Networks
  - https://arxiv.org/abs/1608.06993
- Pre-trained Models using MXNet
  - https://github.com/liuzhuang13/DenseNet
  - DenseNet-121
  - DenseNet-169
  - DenseNet-201
  - DenseNet-161
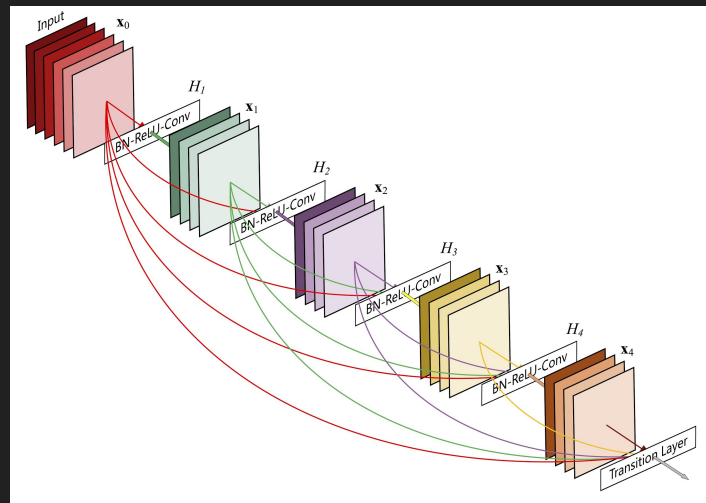
Dual Path Networks

- Paper
  - Dual Path Networks
  - https://arxiv.org/abs/1707.01629
- Pre-trained Models using MXNet
  - https://github.com/cypw/DPNs
  - DPN-68
  - DPN-92
  - DPN-98
  - DPN-131
  - DPN-107
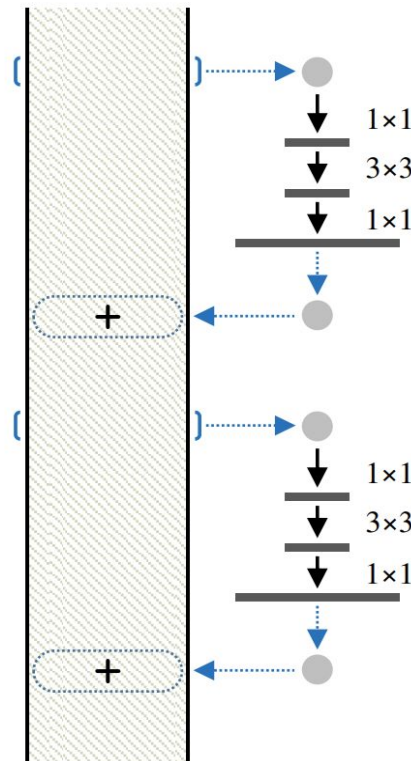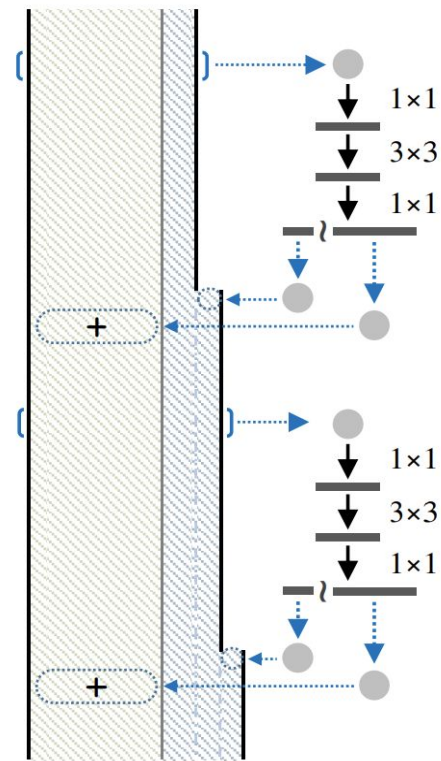
(a) Residual Network   (e) DPN

ResNext

- Paper
  - Aggregated Residual Transformations for Deep Neural Networks
  - https://arxiv.org/abs/1611.05431
- Pre-trained Models using MXNet
  - https://github.com/apache/incubator-m
  - http://data.mxnet.io/models/imagenet/
  - ResNext-50
  - ResNext-101
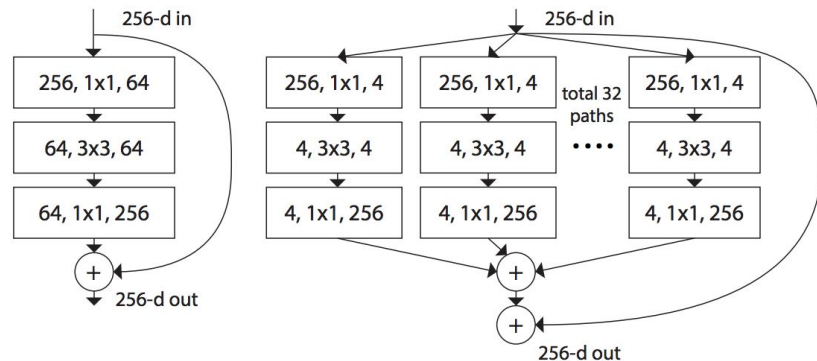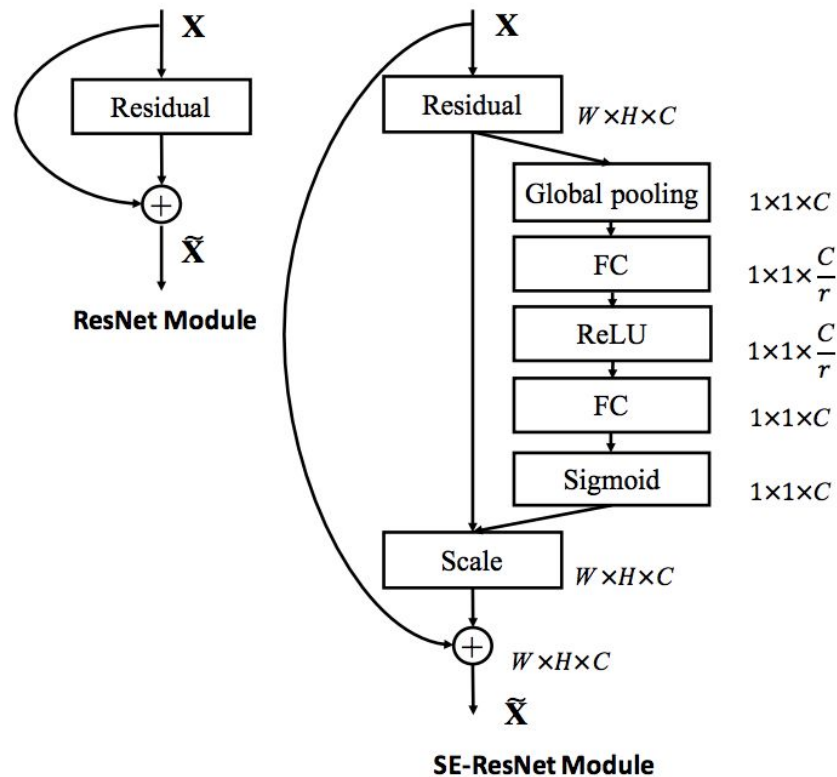 - ResNext-101-64x4d

Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

SE-ResNext

- Paper
  - Squeeze-and-Excitation Networks
  - https://arxiv.org/abs/1709.01507
- Pre-trained Models using MXNet
  - Not Exists
- Implementation
  - I modified the ResNext
    and implemented it myself.

Training Speeds

	Batch Size	GPUs	Memory Usage (per GPU)	Speed (8-GPU, Sync)
ResNext-50	512	8	N/A	N/A
ResNext-101-32x4d	512	8	12,461MiB	579.86 / sec
ResNext-101-64x4d	512	8	18,185 MiB	338.29 / sec
SE-ResNext-101-32x4d	512	8	14,057 MiB	411.55 / sec
SE-ResNext-101-64x4d	512	8	19,805 MiB	274.04 / sec
DPNs-92	512	8	N/A	N/A
DPNs-98	512	8	16,367 MiB	380.22 / sec
DPNs-107	256	8	17,707 MiB	246.06 / sec

Training

Parameter Initialization

- Transfer Learning from ImageNet Models
  - used pre-trained models as much as possible.
- MSRA PReLu Initializer
  - This initializer is proposed for initialization related to ReLu activation, it maked some changes on top of Xavier method.
  - it used last layer for classification
  - https://mxnet.incubator.apache.org/api/python/optimization/optimization.html#mxnet.initializer.MSRAPrelu
  - Introduced in https://arxiv.org/abs/1502.01852

Optimizer

- SGD
- NAG
- Adam
- NAdam → It's the best in this competition.

Augmentation

- input: 180x180x3
- did not use random crop
- use only random flip (it's enough for training on 15~20 epochs)
- experiment
  - ResNet-34, Batch=512, Opt=Adam, Base-LR=0.0005, Epoch=15

Input Size	Option	Local Train (Top-1)	Local Validation (Top-1)
180 x 180	No Aug.	0.862802	0.688090
180 x 180	**Flip**	**0.787349**	**0.692491**
180 x 180	Flip + HSL	0.774842	0.690422
224 x 224 (NN)	Flip	**0.789405**	**0.695116**

Image Size

- The larger the image size, the higher the accuracy.
  - used 180x180 because the larger the image size, the slower the learning.
- Experiment
  - ResNet-34, Batch=512, Opt=Adam, Base-LR=0.0005, Epoch=15

Input Size	conv0	stage1	stage2	stage3	stage4	Train-Acc	Val-Acc	Training Speed
160 x 160	80 x 80	40 x 40	20 x 20	10 x 10	5 x 5	0.753975	0.678730	1130/sec
180 x 180	90 x 90	45 x 45	23 x 23	12 x 12	6 x 6	0.750308	0.680361	872/sec
192 x 192	96 x 96	48 x 48	24 x 24	12 x 12	6 x 6	**0.751713**	0.680706	884/sec
224 x 224	112 x 112	56 x 56	28 x 28	14 x 14	7 x 7	0.748252	**0.681723**	647/sec

Label Smoothing

- https://arxiv.org/abs/1512.00567 (see Section. 7)
- Label smoothing(0.1) increases accuracy.
- Experiment
  - ResNet-34, Batch=512, Opt=Adam, Base-LR=0.0005, Epoch=15

Smooth alpha	Local Train (Top-1)	Local Validation (Top-1)
0.0	**0.785019**	0.691917
0.1	0.779468	**0.694035**
0.2	0.770593	0.692766

GAP + Dropout

- did not use dropout after GAP(Global Average Pooling) layer in such as ResNext, SE-ResNext
- Experiment
  - ResNet-34, Batch=512, Opt=NAdam, Base-LR=0.0005, Epoch=15

Dropout Ratio	Local Train (Top-1)	Local Validation (Top-1)
0.0	**0.788686**	**0.693604**
0.2	0.743764	0.689954

Trained 14 Models

	Network	Transfer from	Dataset	Epochs	Single Val-acc
M01	ResNext-101	ImageNet-1K	C	23	0.660329
M02	DPNs-92	ImageNet-1K	C	13	0.662091
M03	DPNs-92	ImageNet-1K	C	13	0.663739
M04	DenseNet-161	ImageNet-1K	A	20	0.726930
M05	DPNs-98	ImageNet-1K	A	20	0.732641
M06	DPNs-92	ImageNet-1K	A	11	0.734462
M07	ResNext-101-64x4d	ImageNet-1K	A	15	0.735808

Trained 14 Models

	Network	Transfer from	Dataset	Epochs	Single Val-acc
M08	DPNs-131	ImageNet-1K	A	18	0.736697
M09	ResNext-101-64x4d	ImageNet-1K	B	15	0.737427
M10	ResNext-101	ImageNet-1K	A	19	0.738542
M11	SE-ResNext-101-64x4d	M10	A	15	0.739272
M12	DPNs-131	ImageNet-1K	B	17	0.742388
M13	SE-ResNext-101-64x4d	M09	B	13	0.743221
M14	DPNs-107	ImageNet-1K	A	20	0.743781

Prediction

Testing Time Augmentation

- use original image and flipped image
  more augmentation(crop, flop, transpose) reduces accuracy.
- compute arithmetic mean for the probability of a image

Ensembles

- Ensemble of images in a product
  - use arithmetic mean (sum of probabilities of each images)

- Ensemble of models
  - use arithmetic mean (sum of probabilities of each models)

- number of inferences for single image
  - M: number of models for ensemble
  - K: TTA(Testing Time Augmentation)
  - need (M * K) inferences to compute the probability of single image

MD5 Hash

- Create a dictionary of MD5 Hash
  - compute the MD5 hash of each images in the training set.
  - compute the MD5 hash of each products in the training set.
    the hash of the product is a set of md5 hash of each images in the same product.
  - create a dictionary of (hash, category) pairs
  - same image/product may have multiple category. (confused...)

MD5 Hash

- Predict a probability of images using MD5 dictionary
  - during inference time,
    if the hash of a image exists in the dictionary,
    the probability of a image is set to one-hot vector.
  - after using this method,
    - 0.77100 → 0.77429 (+0.00329, Private LB)
    - 0.75890 → 0.76259 (+0.00369, Private LB)
    - 0.75604 → 0.76083 (+0.00479, Private LB)

MD5 Hash

- if same image/product have multiple category → confused...
  - See `predict/predict.py#L133-L161` for details
- Experiment
  - ResNext-101-64x4d (epoch 15, original+flip)

Mode	val-acc
None	0.758480
Unique	0.761724
Majority	0.757597
L1	0.761747
L2	0.761574
Softmax	0.753390

MD5 Hash

- Post-processing to improve accuracy using MD5 hash of products
    - after inference time,
      if the hash of a product exists in the dictionary,
      category of this product is overwritten by the category in the dictionary.
    - after using this method,
        - 0.78998 -> 0.79046 (+0.00048, Private LB)
   (it's BEST SCORE for me)

Accelerate Pre-processing

- needs multi-processing for fast pre-processing but, `multiprocessing.Queue` module is very slow.
- I use ZeroMQ instead of `multiprocessing.Queue` for process communication.

Leaderboard with Ensembles

Ensembles	Models	Private LB	Public LB
1	M14	0.76713	0.76544
2	M5, M7	0.77429	0.77282
4	M11~M14	0.78693	0.78490
8	M06~M14	0.78775	0.78577
11	M04~M14	0.78879	0.78673
14	M01~M14	0.78998	0.78813
14	M01~M14 (with post-processing using MD5)	0.79046	0.78861

3rd place

#	△pub	Team Name	Kernel	Team Members	Score	Entries
1	—	**bestfitting**			0.79567	63
2	—	**Convoluted predictions**			0.79352	155
3	—	**Dylan @ Kakao MMRDT**			0.79046	90
4	—	**Dimensionality Diabolists**			0.78868	95
5	—	**10011000**			0.78693	169
6	—	**smlyaka**			0.78315	96
7	—	**DeepTortoise**			0.78178	77
8	—	**Azat Davletshin**			0.77883	12
9	—	**n01z3**			0.77735	51
10	▲1	**owruby**			0.77611	61
11	▼1	**Guanshuo Xu**			0.77604	6

Other Methods

Large Margin Softmax
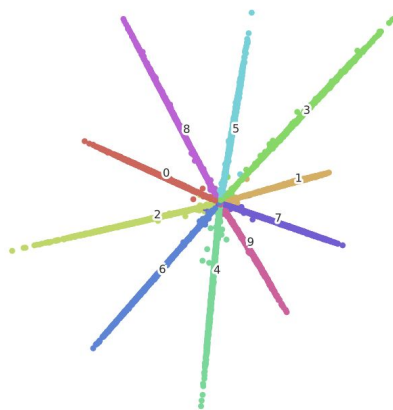
- https://arxiv.org/abs/1612.02295
- MXNet implementation: https://github.com/luoyetx/mx-lsoftmax
- training failed... :(

original softmax (traditional fully connected)
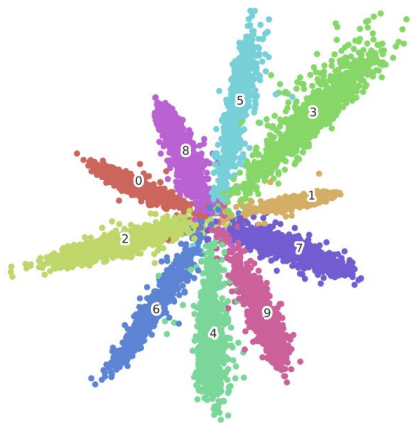
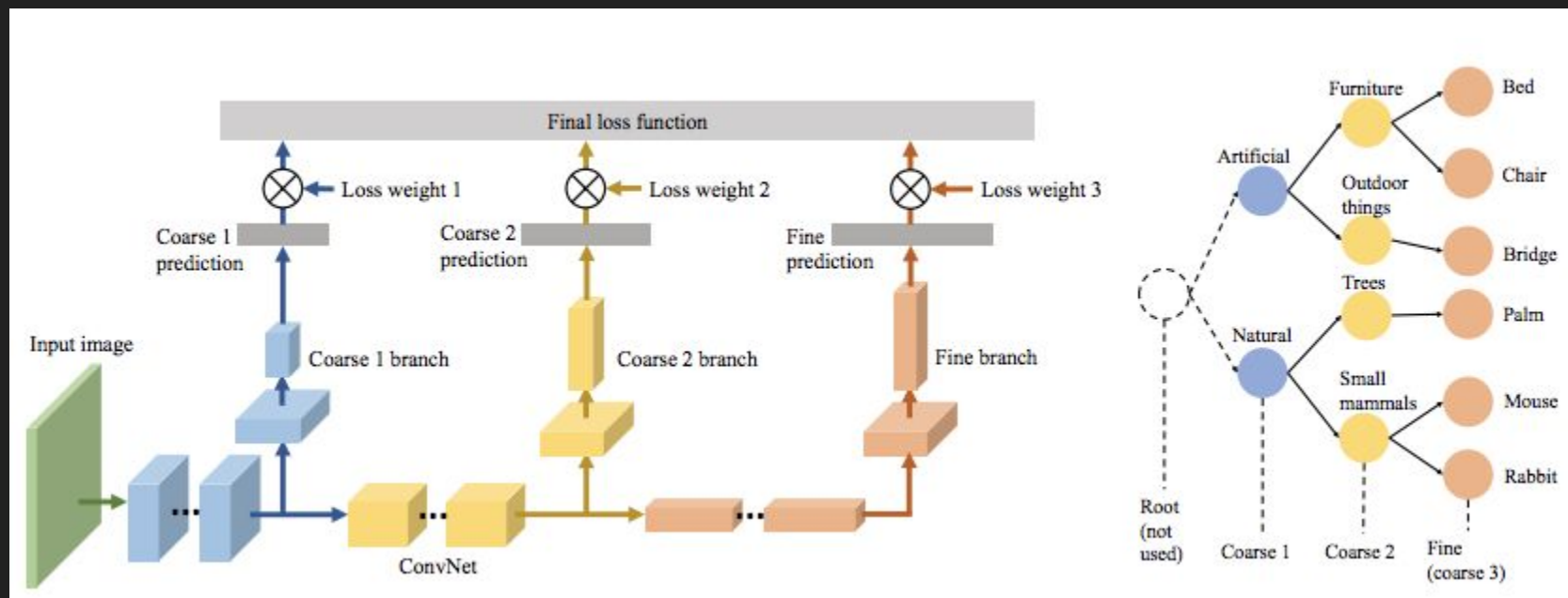

lsoftmax with margin = 6, beta = 1000 and scale = 0.99

OCR

- Tried Tesseract OCR to improve accuracy for Book and Album
  - https://github.com/tesseract-ocr/tesseract
  - I tested some sample images
  - but, The OCR results were not accurate

B-CNN

- https://arxiv.org/abs/1709.09890
  - "a good initialization may downgrade the benefit of B-CNN"

12-Channel CNN

- Combine four images to create a 12-channel image.
- I did not have enough time for training :(

ILSVRC 2017 Classification Competition

A. The number of first 1×1 convolutional channels for each bottleneck building block was halved to reduce the computation cost of the network with a minimal decrease in performance
B. The first 7 × 7 convolutional layer was replaced with three consecutive 3×3 convolutional layers.
C. The down-sampling projection 1 × 1 with stride-2 convolution was replaced with a 3 × 3 stride-2 convolution to preserve information

ILSVRC 2017 Classification Competition

D.   A dropout layer (with a drop ratio of 0.2) was inserted before the classifier layer to prevent over-fitting
E.   Label-smoothing regularisation was used during training.
F.   The parameters of all BN layers were frozen for the last few training epochs to ensure consistency between training and testing

https://arxiv.org/pdf/1709.01507.pdf

References

- ILSVRC 2017
  - http://image-net.org/challenges/beyond_ilsvrc
  - SE-Nets: http://image-net.org/challenges/talks_2017/SENet.pdf
  - DPNs: http://image-net.org/challenges/talks_2017/ilsvrc2017_DPNs.pdf
  - Short: http://image-net.org/challenges/talks_2017/ilsvrc2017_short(poster).pdf
- ILSVRC 2016
  - Hikvision ImageNet 2016:
    http://image-net.org/challenges/talks/2016/Hikvision_at_ImageNet_2016.pdf