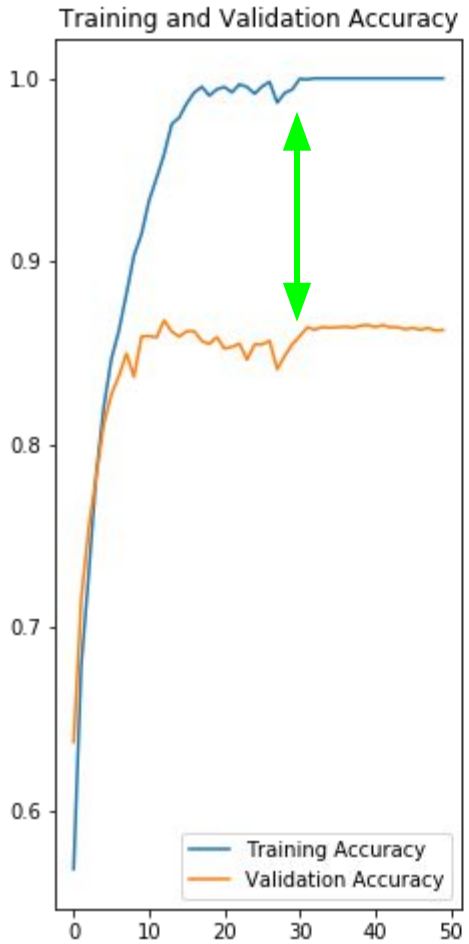
Learning Deep Learning with PyTorch

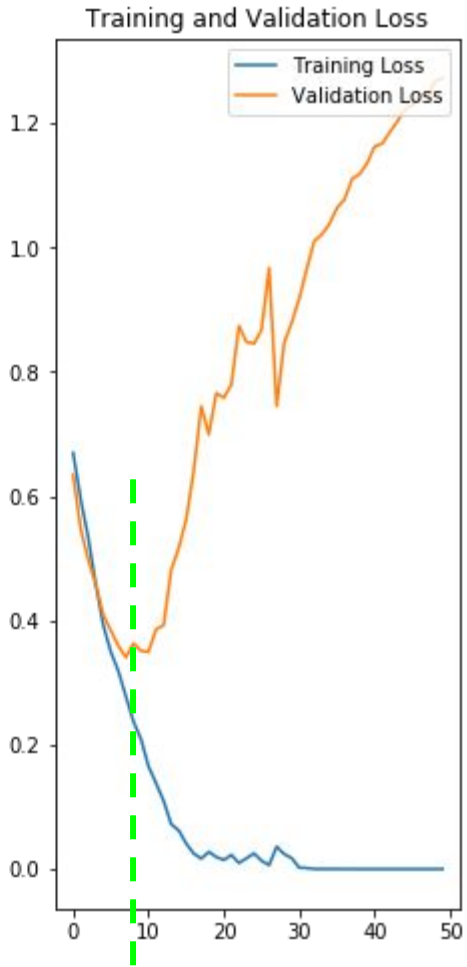
(5) Data Augmentation & Transfer Learning

Qiyang Hu
IDRE
May 4th, 2020

Quick Recap

- Dogs-vs-Cats challenges
  - 25,000 training images
  - 15,000 testing images
- Construct our own CNNs
  - 4 Conv layer blocks
  - Flatten layer
  - Dense layer
- Overfitting
  - Memorizing training set too much
  - Missing the essence knowledge
- How to improve?
  - Need more training data
  - Need regularization



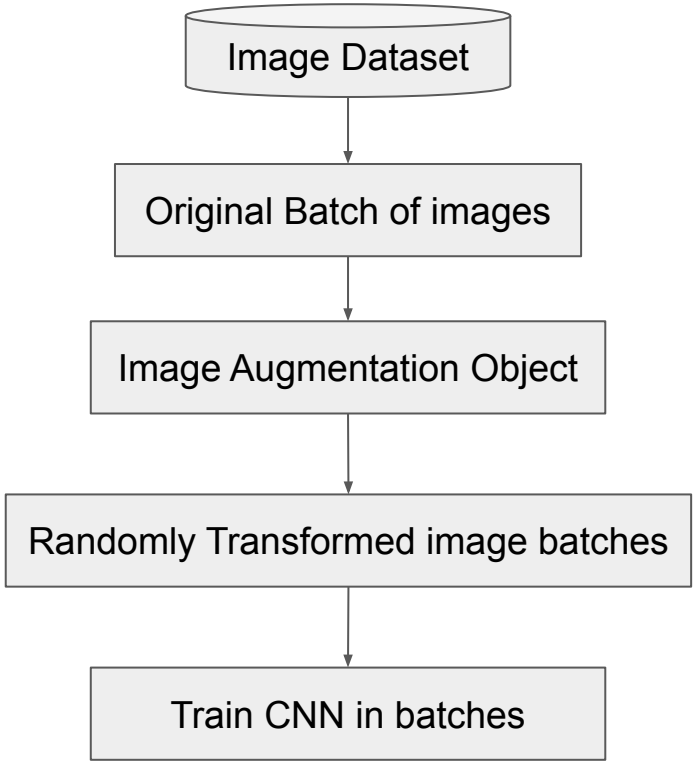


Dataset: the bigger the better, but why?

	VGGNet	DeepVideo	GNMT	GPT-2
Used For	Identifying Image Category	Identifying Video Category	Translation	Text Generation
Input	Image	Video	English Text	Text
Output	1000 Categories	47 Categories	French Text	Text
Parameters	140M	~100M	380M	1.3B
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words	8M Web pages

How to get more data with “no more”?

- Use data augmentation
  - Various transformations to the available dataset
  - Prevent the irrelevant data
- Types of data augmentation
  - Offline augmentation
    - Performing all the transformations beforehand
    - Good for smaller dataset
  - In-place augmentation
    - Performing transformations in mini-batches
    - Preferred for larger dataset
- Data augmentation in PyTorch
  - torchvision.transforms

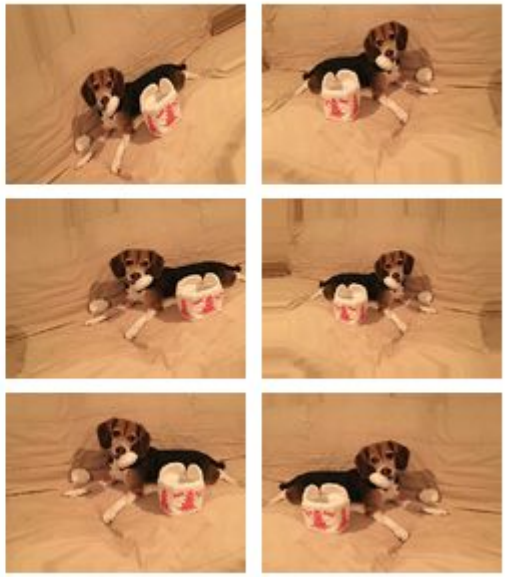


Augmentation Techniques

- Flip
- Affine Transformation
  - Rotation
  - Zoom & Crop
  - Translation
- Gaussian Noise
- ZCA whitening
- Histogram Equalization
- Feature-wise standardization
- Neural Style Transfer (cGANs)

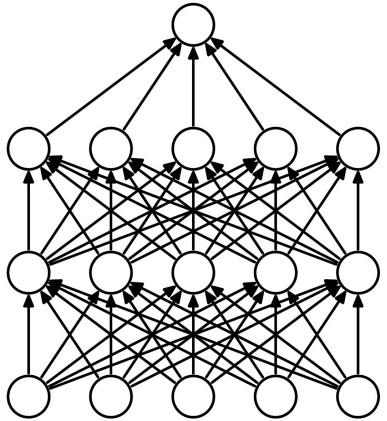
Input Image

Augmented Images

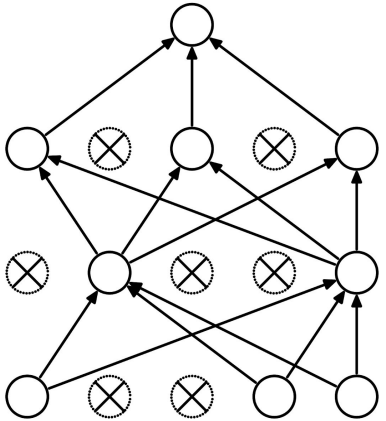


Drop-out technique

- Gradient vanishing during DNN training:
  - Imbalanced weights in network:
    - Larger weights => well trained
    - Smaller weights => not trained that much!
- Dropout: randomly turns off some neurons
  - Forcing networks to train weak neurons
  - Dropout rate: default 50%
    - Roughly double the iterations to converge
      Training time in epoch is less
  - From Srivastava 2014 paper:
- PyTorch: torch.nn.Dropout2d()
  - Apply to the corresponding layer(s)



(a) Standard Neural Net



(b) After applying dropout.

Other training techniques in deep learning

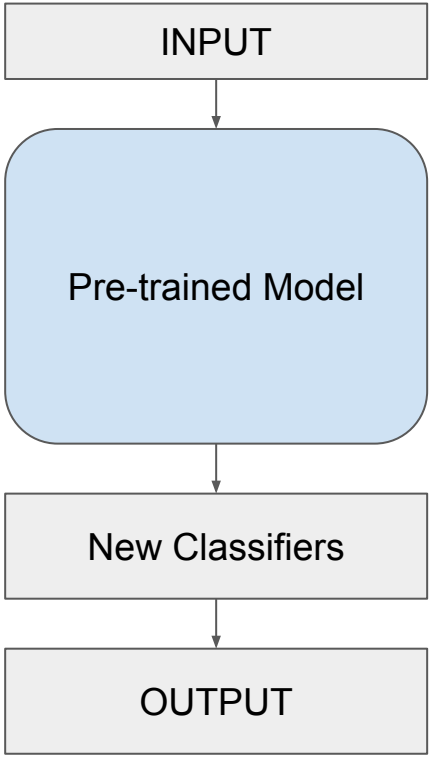
- Regularizer
  - l1(Lasso), l2(Ridge), l1_l2(ElasticNet) in each layer
    - `weight_decay` flag for l2 in pytorch optimizers
- Early Stopping
  - Stop training when validation loss reach minimum
- Batch Normalization (may mutually exclude drop-out, see paper)
  - Define layer without bias
  - Add batch normalization before activation function
    - `torch.nn.BatchNorm2D(num_features=n_chans1)`
- Skip connections
  - Simple trick to add the input (conv1) to the output of a block of layers (conv3)
  - Residual networks (K. He, 2015)
    - Opened the door to hundreds-layer-depth networks (Highway Net, U-Net, Dense-Nets, …)

Colab Hands-on

bit.ly/LDL_03

Transfer Learning

- Reusing the developed neural networks
  - Greatly speed up our training
  - Make it mobile
- Image classification
  - Advanced models from ImageNet competition
    - AlexNet, ResNet, Inception v3, …
  - torchvision.models
    - resnet = models.resnet152(pretrained=True)
- Simple steps
  - Match the input size of images from the pre-trained model.
  - Define our new classifiers
    - ImageNet classes: 1280; Our classes: 2



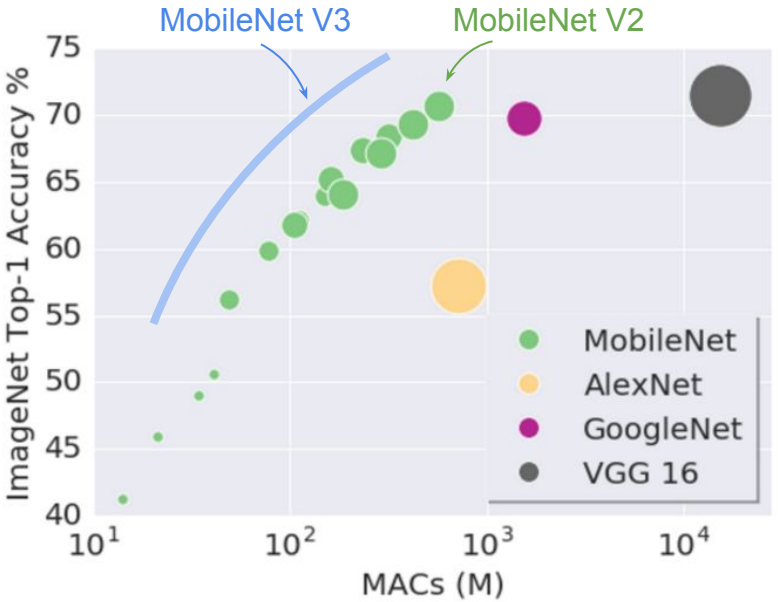
MobileNet v2 & v3

- Very efficient CNNs (v2 paper & v3 paper)
  - Especially good for mobile vision apps
- PyTorch Hub
  - Pretrained models from the latest research
  - Published through GitHub
    - Check hubconf.py
- Loading the model from Hub

```
model = torch.hub.load( 'pytorch/vision:v0.5.0',
                        'mobilenet_v2',
                        pretrained=True
                        )
```

- Modify the classifier layer

```
model.classifier[1] = torch.nn.Linear(1280, 2)
```

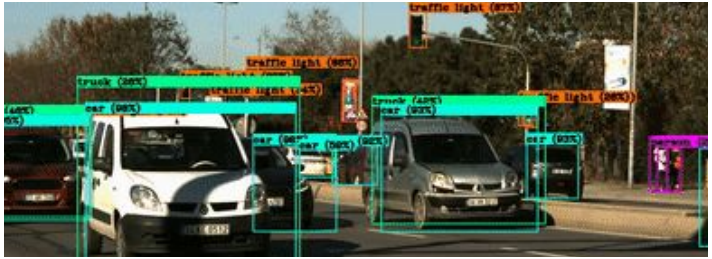


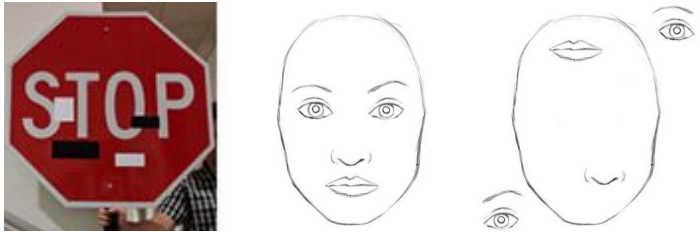
(MACs = Multiply-Accumulates)

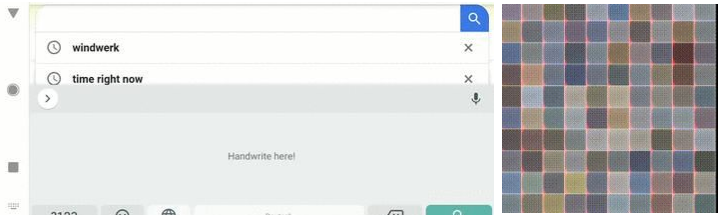
Figure from v2 paper and v3 paper

CV Topics Beyond

- Classifying from images to regions
  - R-CNN, Fast R-CNN, Faster R-CNN
  - Mask R-CNN, R-FCN, SSD, RetinaNet
  - YOLOv4, Detectron2, ...
- Improving classification:
  - Adversarial Attacks
  - Capsule Networks
    - spatial hierarchies to inverse graphics
- Recurrent Neural Networks
  - Dynamic input and output layers
  - Especially effective in sequence problems







FYI

- Github Repo:
  - https://github.com/huqy/idre-learning-deep-learning-pytorch
- Slack workspace:
  - bit.ly/join-LDL
- Contact me
  - huqy@idre.ucla.edu
  - Direct message in Slack
- IF you don’t have plan to attend the last of workshops,:
  - Please fill out our survey for the whole series: bit.ly/2X2phyS