Transfer Learning

Eu Wern Teh

What are we covering?

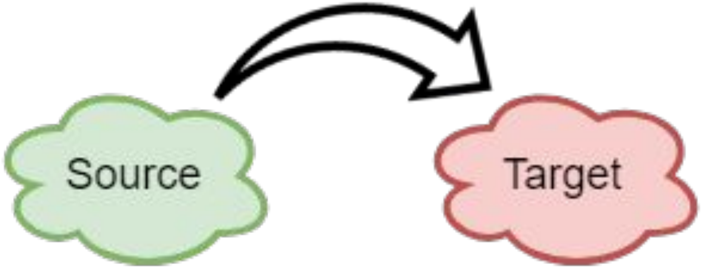
- Why transfer learning?
- Fine Tuning: how? why? Example
- Practise:
  - Ants and Bees dataset
  - Caltech 101 dataset

Why Transfer Learning?

- Speedup Training
- Improve generalization (especially: less data)
- Transfer Learning:
    - Fine Tuning
    - Domain Adaptation
        - train a model that does the same thing on different environment
            - eg: object classification on high vs low resolution image
            - eg: customer rating estimation on various domain (travel review vs hotel review)

Domain Adaptation

- Source and Target domain have the same labels but in a different domain
- Large number of labeled data source domain, but opposite for the target domain

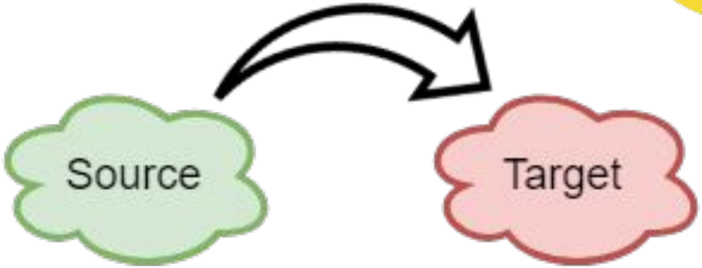


Amazon

Webcam

Domain Adaptation

Product Rating

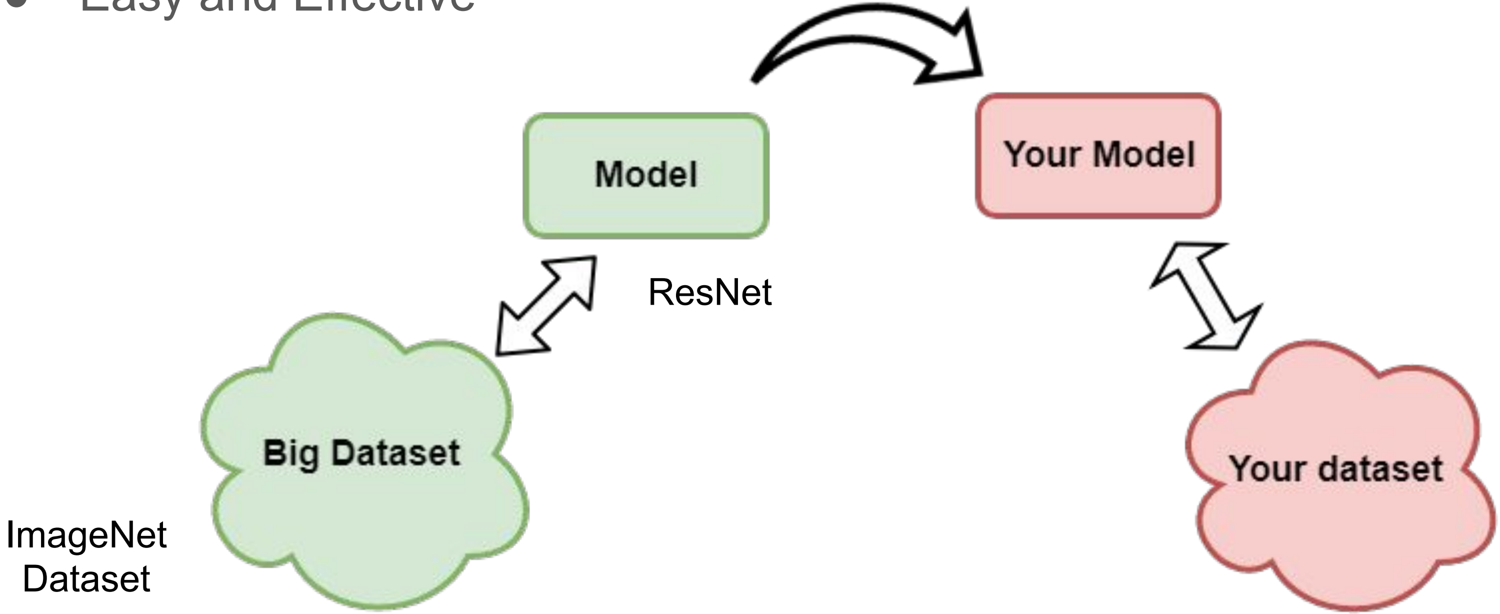


Electronics

Video Games

Fine Tuning

- Most common form of transfer learning.
- Easy and Effective

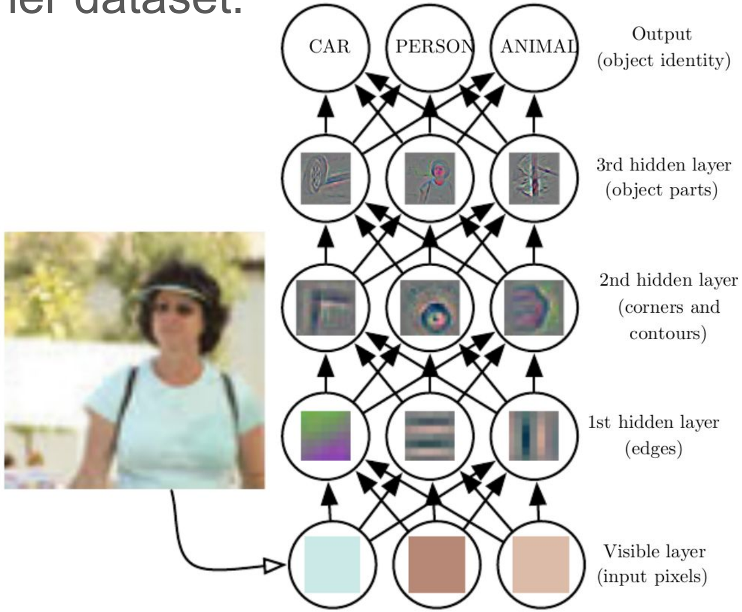


Fine Tuning

- In practise, very few people train their model from scratch (with random weight initialization)
- Fine Tuning is achieved by initializing your model with weights train on another dataset:
  - ImageNet 2012 - 1000 classes and 1 millions images (1000 images per class)
  - VGG Faces dataset - 2622 identities and 2.6 millions images (991 images per class)
- Learning rate manipulation during training
  - high vs low learning rate will affect the testing performance of your new dataset.
  - highly dependent on the size of dataset.

Why fine tuning works?

- Features or Pattern that are working for one dataset may be useful on some other dataset.



A series of stack layers that extracts increasingly abstract features from the image.

The higher the layers, the more abstract the features.

Lines → Edges → Shapes → Object Parts → ...

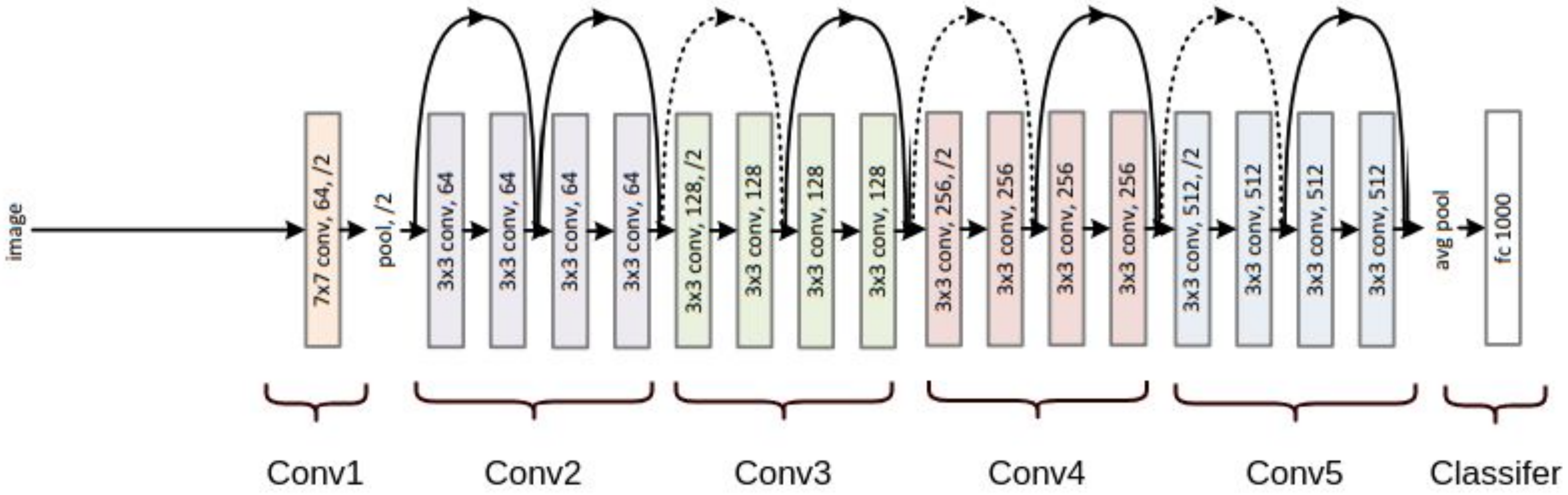
Deep Network

- ResNet architectures:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Deep Network

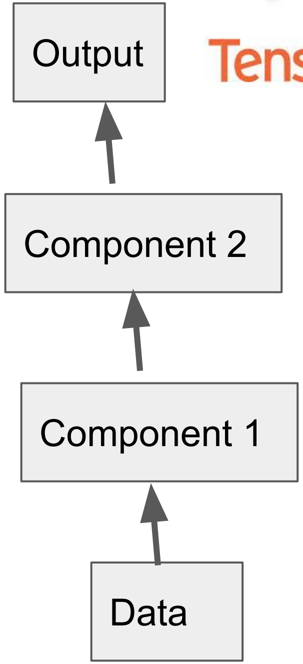
- ResNet-18



Machine Learning Practitioner

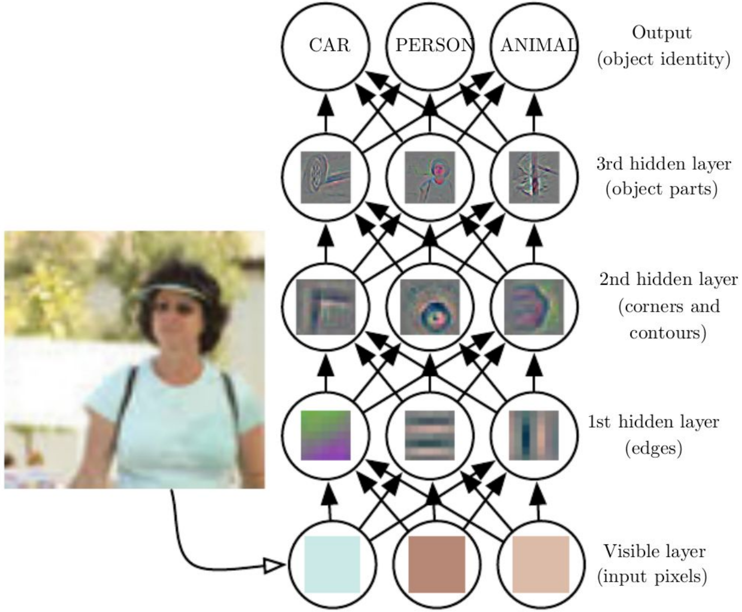
PYTORCH

TensorFlow



Manipulate Learning Rate

- What features should we keep the most? (little to no change)
  what features should we adapt the most? (lots of change → adapt to dataset)
- Lines?
  Edges?
  Shapes?
  Object Parts?
  Classifier?

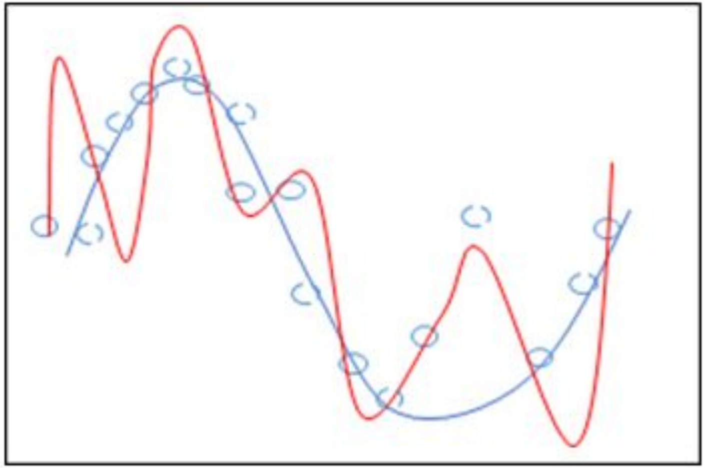


A series of stack layers that extracts increasingly abstract features from the image.

The higher the layers, the more abstract the features.

Lines → Edges → Shapes → Object Parts → ...

Overfitting (Seen Data)



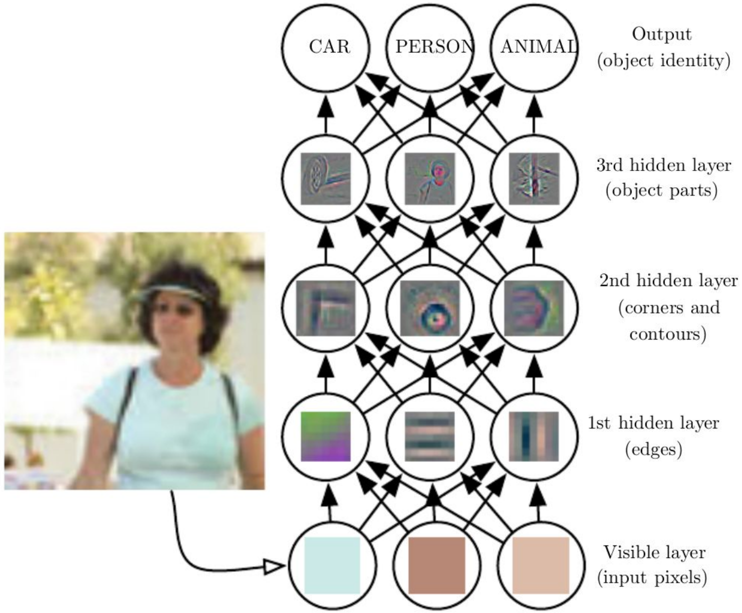
Model A

Generalization
(Unseen Data)

Model B

Manipulate Learning Rate

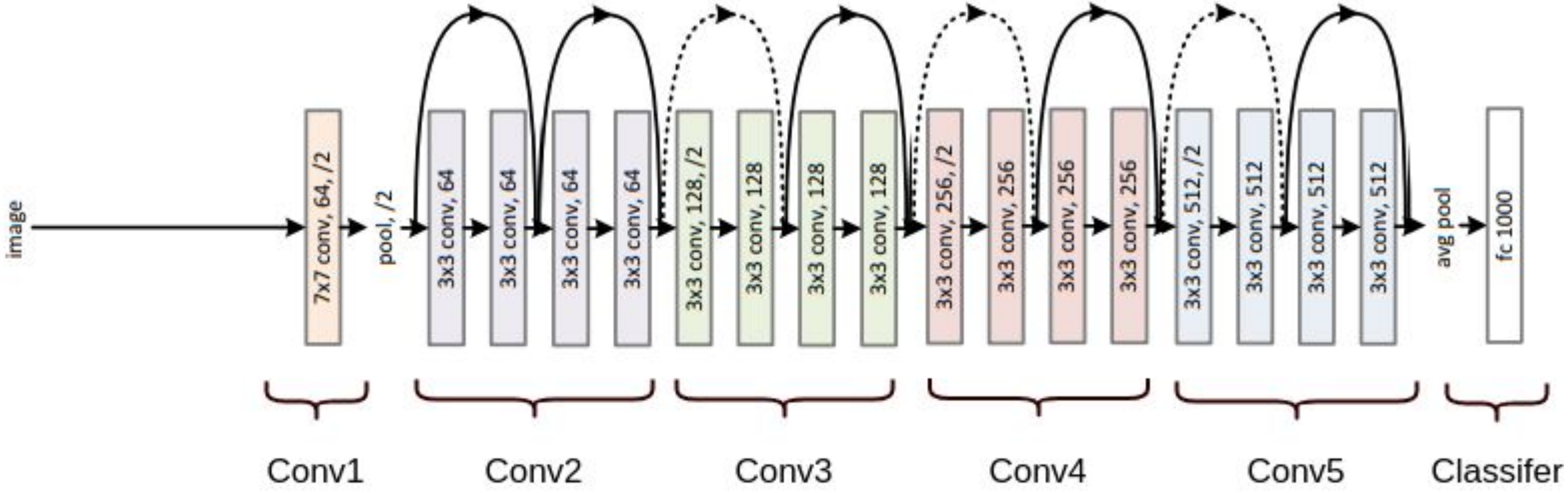
- If we have a small dataset, we would trust the lower level features from our pretrain model. (It has seen more lines, Edges, Shapes ...)
- Lines?
  Edges?
  Shapes?
  Object Parts?
  Classifier?



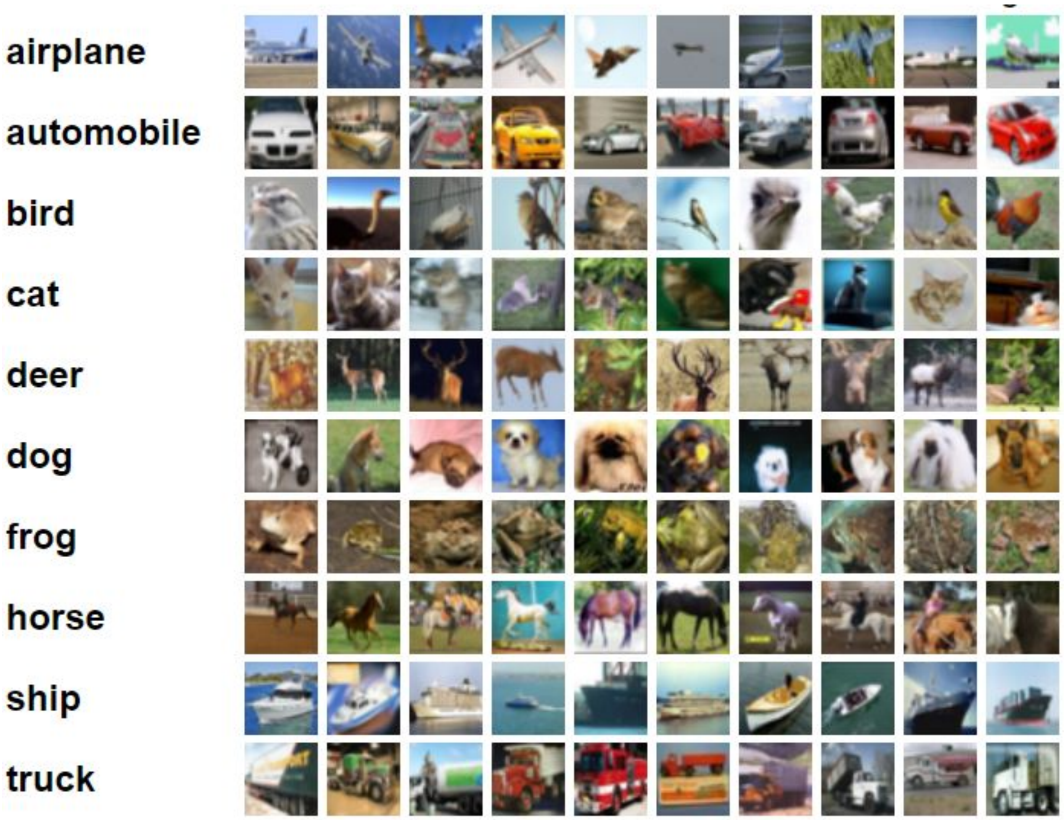
- **We trust it by not changing the features too much → low learning rate**

Deep Network

- ResNet-18



CIFAR 10 dataset



Created by: Alex Krizhevsky, 2009

60,000, 32x32 Color images.

6000 images per class

50,000 training
10,000 testing

Experiments on Fine Tuning on Cifar 10-4000

All models terminate at 100 epochs.

Models	Descriptions	Train Acc	Test Acc
Model A (no fine tuning)	set all lr = 1e-1	99.98%	76.34%
Model B	set all lr = 1e-1	99.61%	66.10%
Model A (no fine tuning)	set all lr = 1e-4	44.97%	41.89%
Model B	set all lr = 1e-4	92.04%	87.14%
Model C	set all features lr = 1e-4 set class lr = 1e-1	99.68%	88.72%

Experiments on Fine Tuning on Cifar 10-4000

All models terminate at 100 epochs.

Models	**Descriptions**	**Train Acc**	**Test Acc**
Model C	set all features lr = 1e-4 set class lr = 1e-1	99.68%	88.72%
Model D	set conv1, conv2 and conv3 lr = 1e-4 set conv4 and conv5 lr = 1e-3 set class lr = 1e-1	99.88%	89.65%

How to do it?

```
opt = optim.SGD([{'params':model.base[0].parameters(), 'lr': args.lr * 1e-3},
                 {'params':model.base[4].parameters(), 'lr': args.lr * 1e-3},
                 {'params':model.base[5].parameters(), 'lr': args.lr * 1e-3},
                 {'params':model.base[6].parameters(), 'lr': args.lr * 1e-2},
                 {'params':model.base[7].parameters(), 'lr': args.lr * 1e-2},
                 {'params':model.fc1.parameters()}], lr=args.lr, momentum=0.9,
                    nesterov=False, weight_decay=5e-4)
```

Freezing Learning Rate

- If you are setting your learning rate to zero on some layers, you should set the requires_grad attribute to False. (This allows you to save some memory and compute time).
    - for param in model.base.parameters():
      param.requires_grad = False

Mean and Standard Deviation adjustment

- Feature Normalization
- you need to normalize your image with the mean and standard deviation of your pre-trained model
- If you fine-tune your network with ResNet model train on ImageNet in Pytorch model:
  - input needs to be scale from 0 to 255 to zero to one
  - the means of RGB: 0.485, 0.456, 0.406
  - the std of RGB: 0.229, 0.224, 0.225
  - https://pytorch.org/docs/stable/torchvision/models.html
- if you fine-tune your network with ResNet model train on ImageNet in Caffe
  - you to load your image in BGR and inputs needs to be in 0 to 255
  - the means of BGR: 103.939, 116.779, 123.68
  - do not need to divide by std.

Ants and Bees dataset

- Small quantity of big resolution images.
- 398 images, 2 classes, (199 images per class)
- 10% training
- 10% validation
- 80% testing
- Random Chance: 50%
- Test Accuracy (base); 60.13%
- Test Accuracy:(fine-tune) 86.71%

https://jupyter.co60.ca

Caltech 101 dataset

- medium quantity of big resolution images.
- 9144 images, 102 classes, (89 images per class)
- 1% training
- 1% validation
- 98% testing
- random chance: ~1%
- Test Accuracy(base): 19.78%
- Test Accuracy(fine-tune): 38.2%


THANK YOU FOR YOUR ATTENTION
ANY QUESTIONS?
memegenerator.net