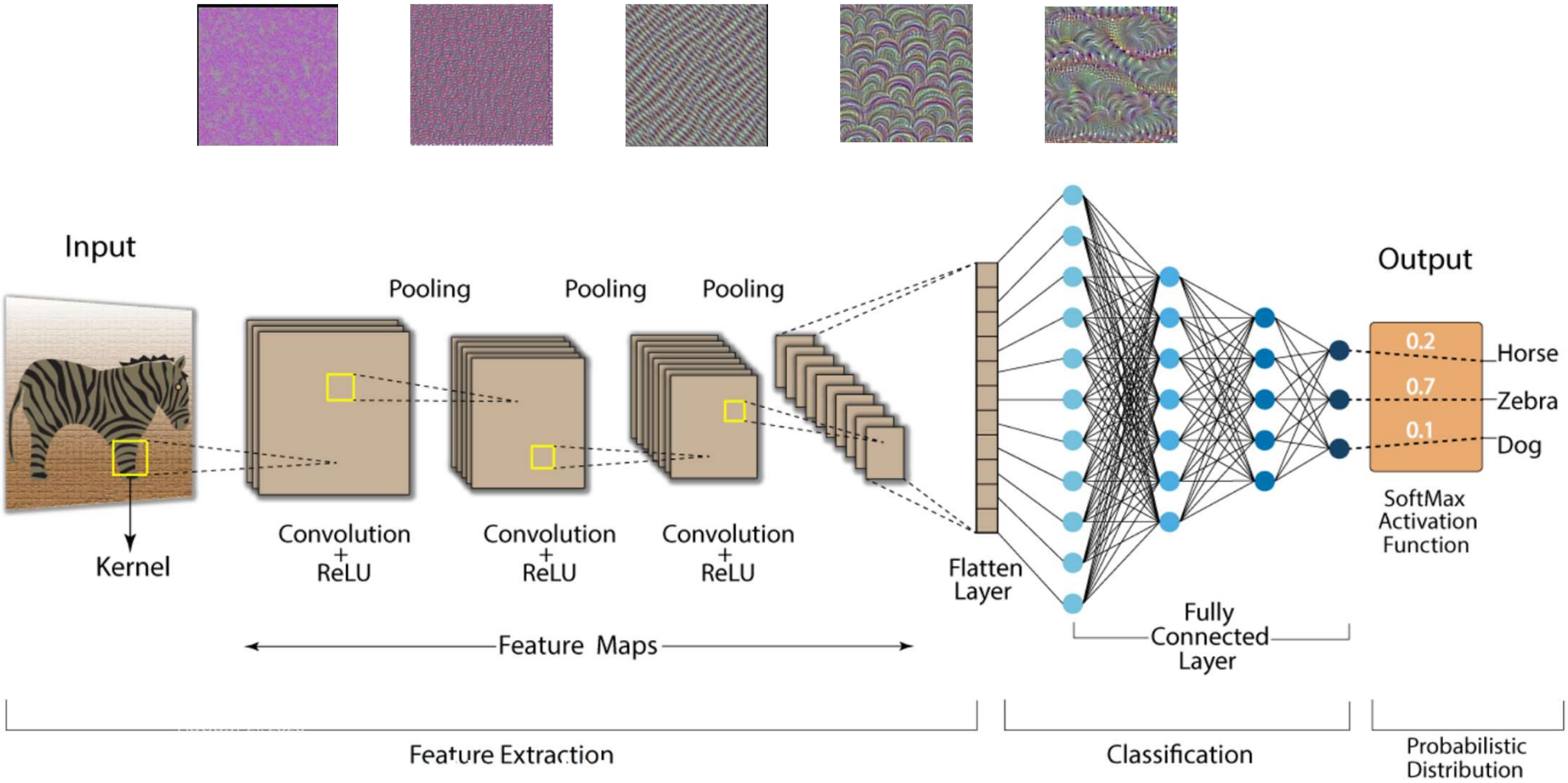
CNN – Quick Recap

Neural Networks with shared weights between layers

and with convolutions

Deep Learning For Image Analysis

- Image data requires subject-matter expertise to extract key features
- Deep learning extracts feature automatically from domain-specific images, without any feature engineering techniques
- Train networks with many layers, with multiple layers working to build an improved feature space
- First layer learns 1st order features (e.g., edges) while 2nd layer learns higher order features
- Lastly, final layer features are fed into supervised layer(s)



Convolution Neural Network Architecture

Demo

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

Loopy pattern filter

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

ReLU
Activation

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max
Pooling

1	1
0.33	0.33
0.33	0.33
0	0

Demo

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

*

Loopy pattern filter

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55

ReLU Activation

1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

Max Pooling

1	0.33
0.33	0.33
0.33	0
0	0

Training CNN with gradient descent

- A CNN as composition of functions

$$f_w(x) = f_L(\dots\ (f_2(f_1(x; w_1); w_2)\ \dots; w_L)$$

- Parameters

$$w = (w_1, w_2, \dots w_L)$$

- Empirical loss function

$$L(w) = \frac{1}{n}\sum_i l(z_i, f_w(x_i))$$

- Gradient descent

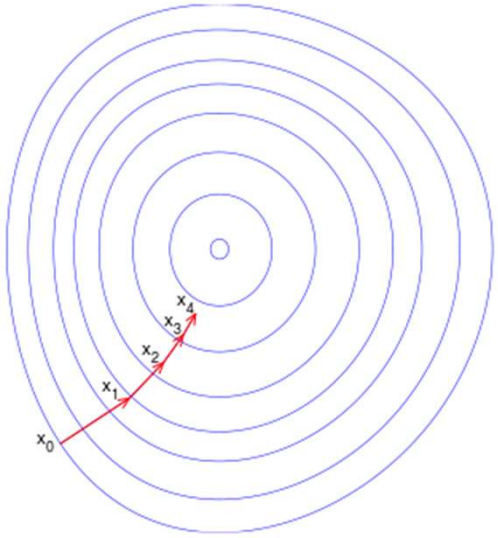
$$w^{t+1} = w^t - \eta_t \frac{\partial L}{\partial w}(w^t)$$

New weight

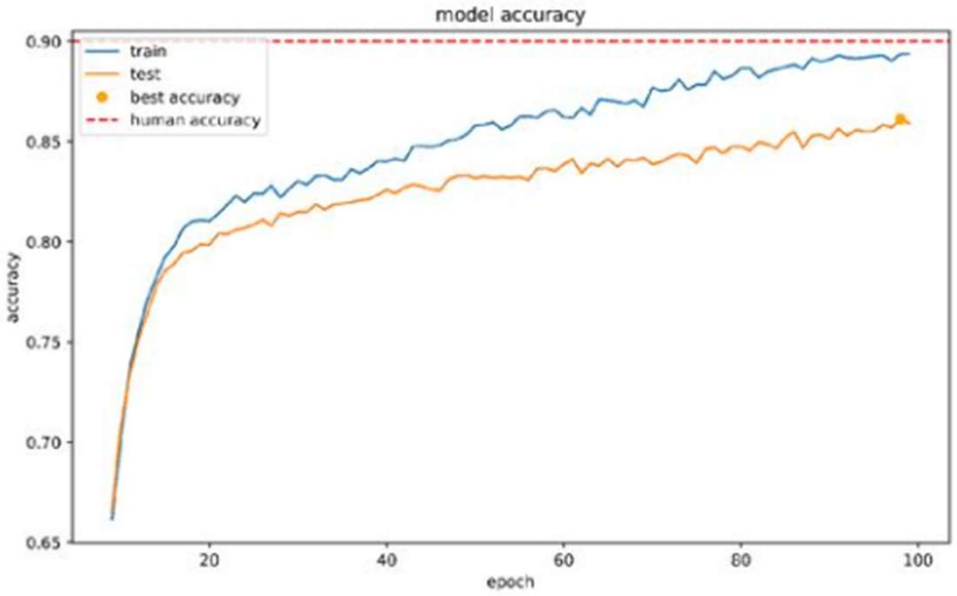
Old weight

Learning rate

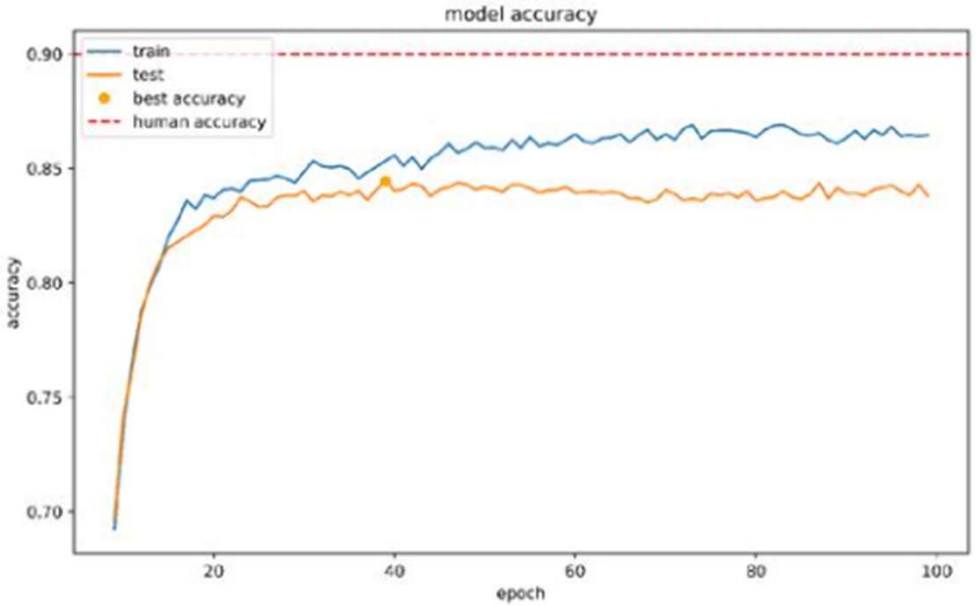
Gradient



Learning Curve

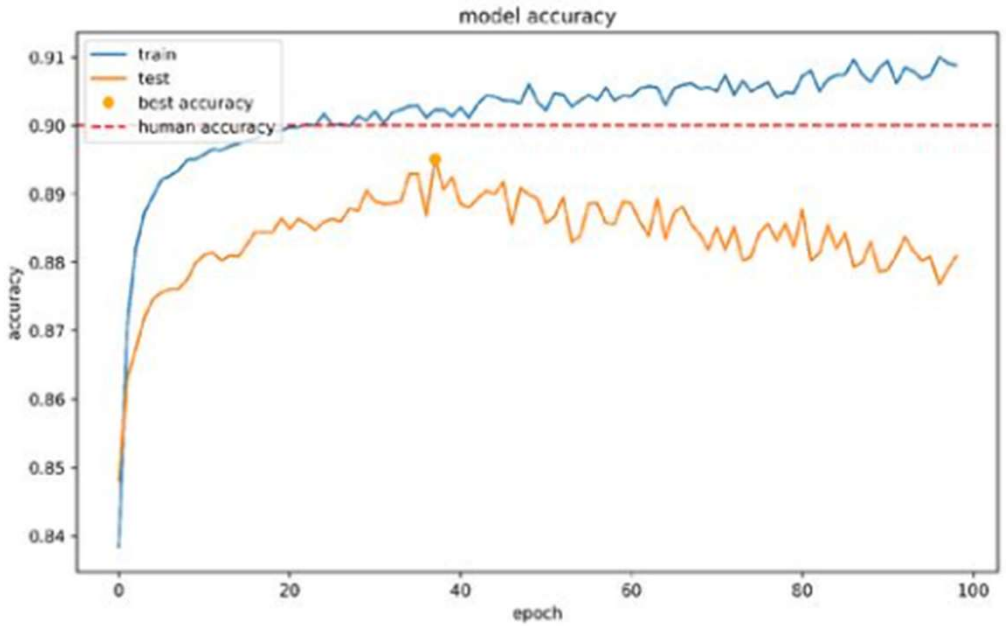


Underfitting: Accuracy may improve with higher learning rate and more epochs

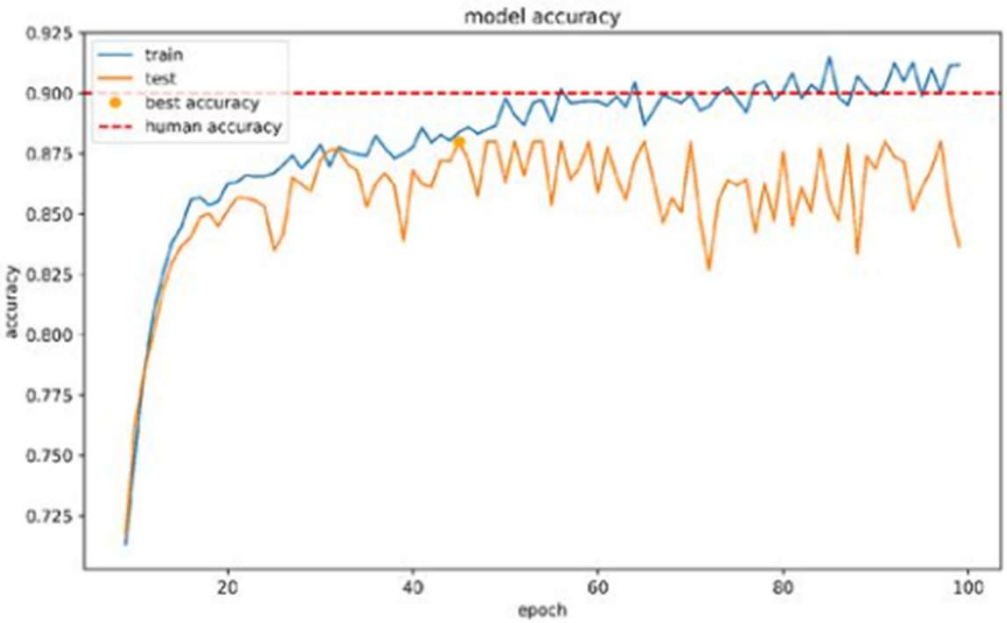


Underfitting: We need a deeper network to improve the accuracy

Learning Curve



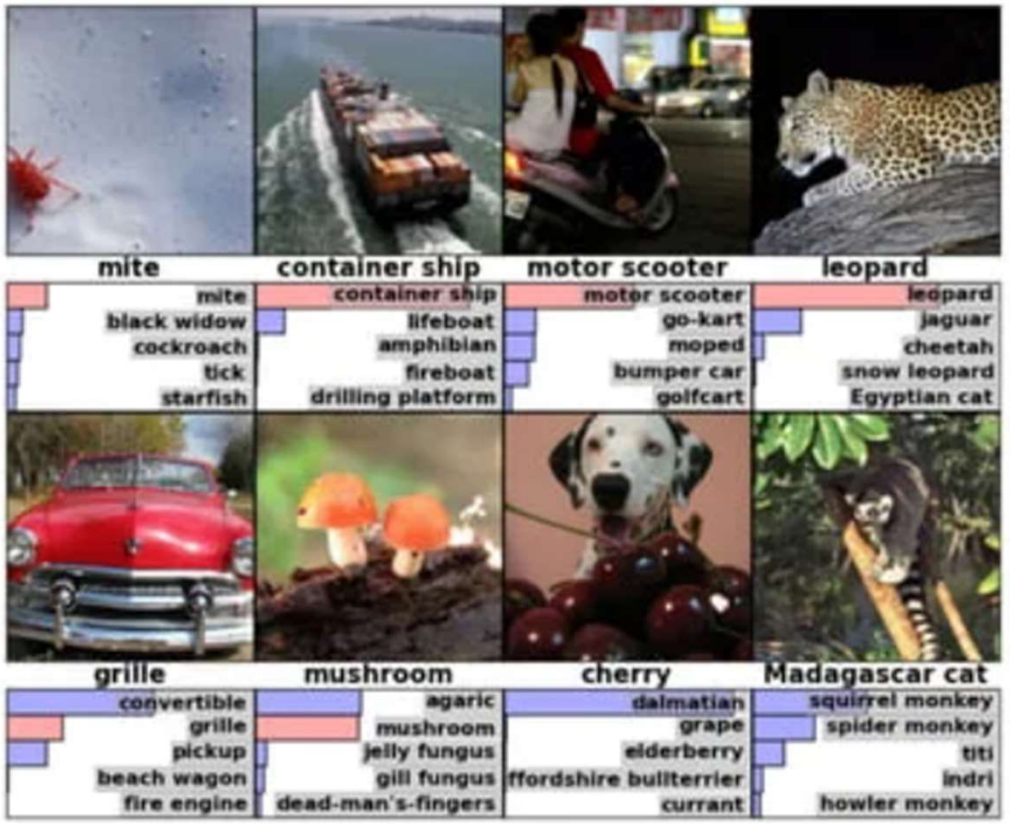
Overfitting: Need to implement regularization techniques or increase the size of dataset



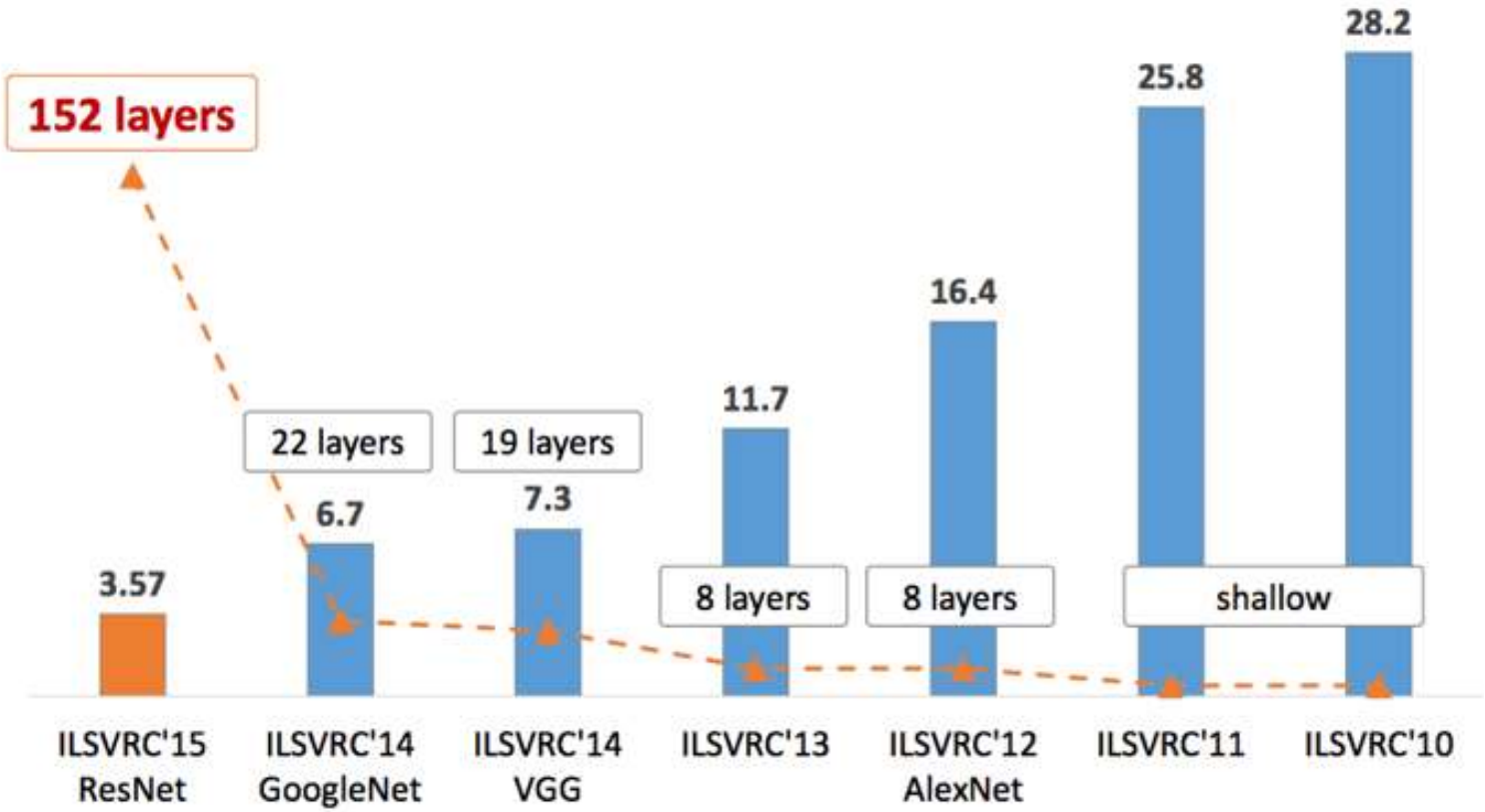
Overfitting with oscillations: Need to decrease the learning rate as the network is becoming unstable after some epochs

ImageNet Challenge

- The goal of this image classification challenge is to train a model that can correctly classify an input image into 1,000 separate object categories.
- Models are trained on ~1.2 million training images with another 50,000 images for validation and 100,000 images for testing.
- These 1,000 image categories represent object classes that we encounter in our day-to-day lives, such as species of dogs, cats, various household objects, vehicle types, and much more.

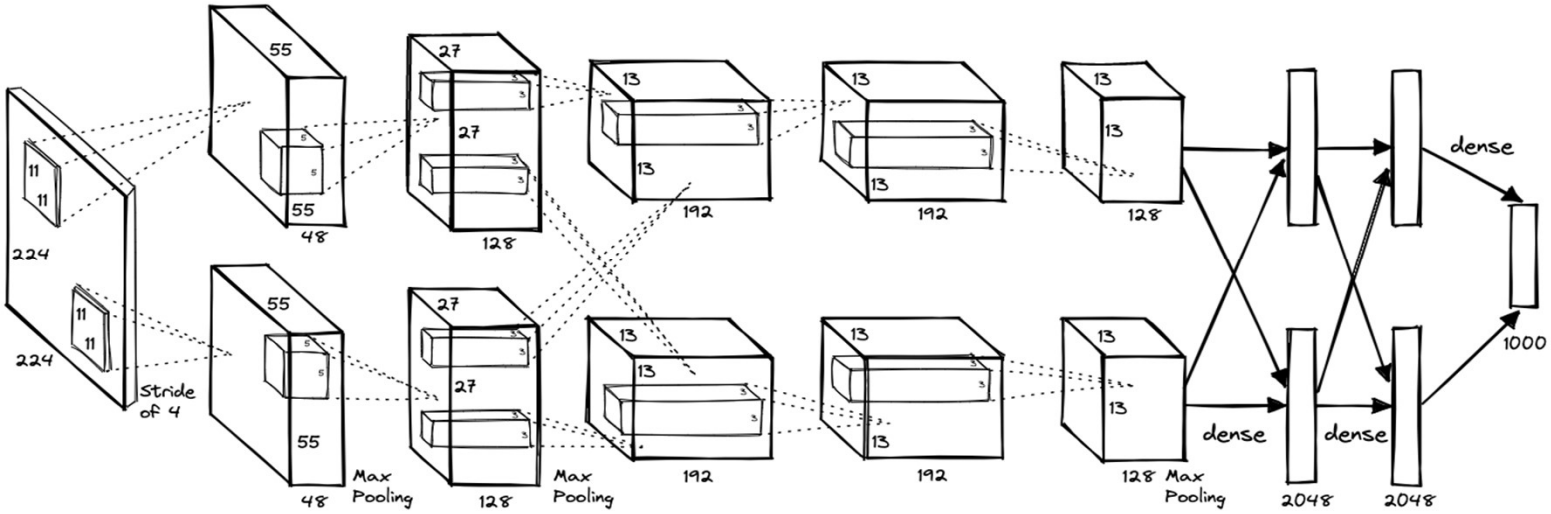


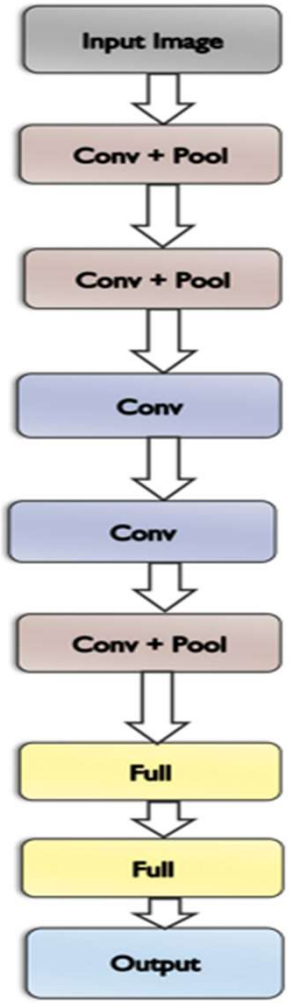
Superhuman object recognition in images



AlexNet

Image credits to Krizhevsky et al., the original authors of the AlexNet paper.



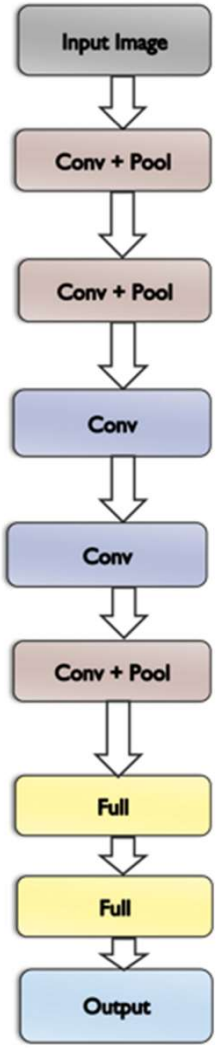


- 5 convolutional layers, 3 max-pooling layers, 2 fully connected layers, and 1 softmax layer.
- ReLU used as an activation function. SGD Momentum used for optimization with batch size of 128

AlexNet's: Overfitting Problem

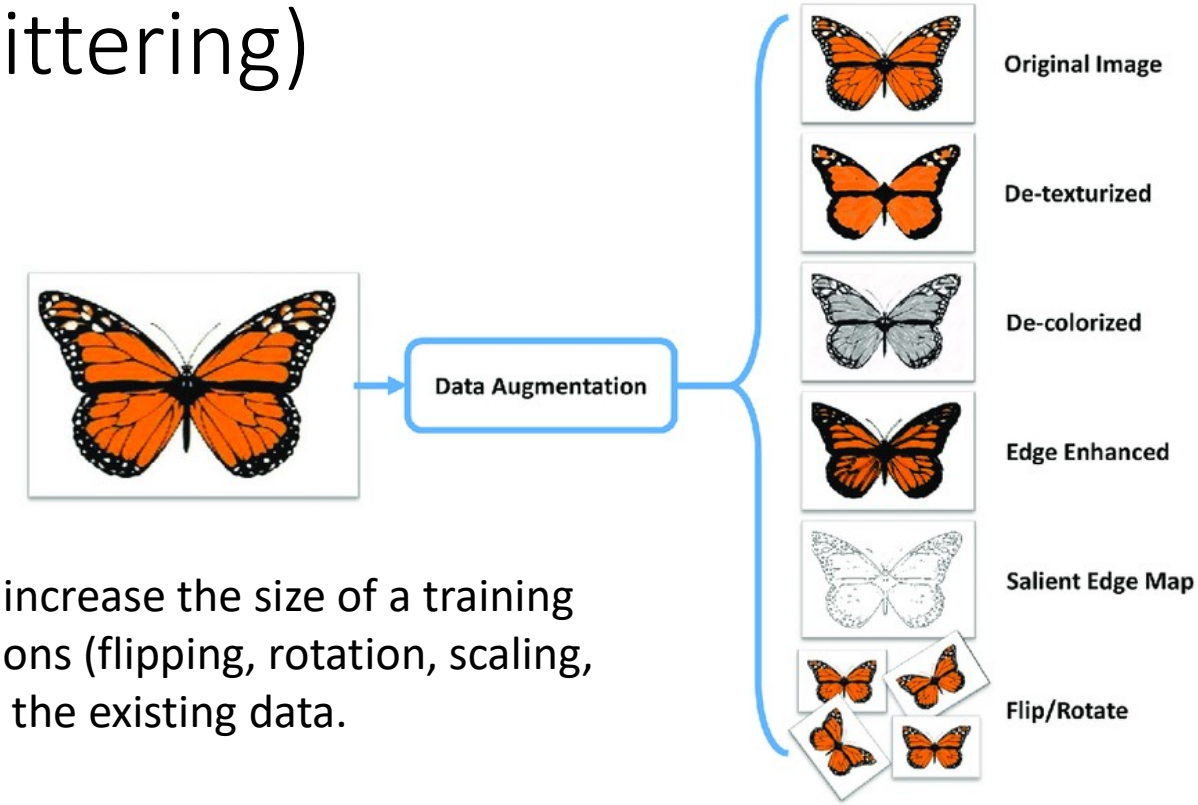
AlexNet had to fit around 60 million parameters, leading to overfitting. They employed two methods to reduce overfitting

- Data Augmentation
- Dropout



Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



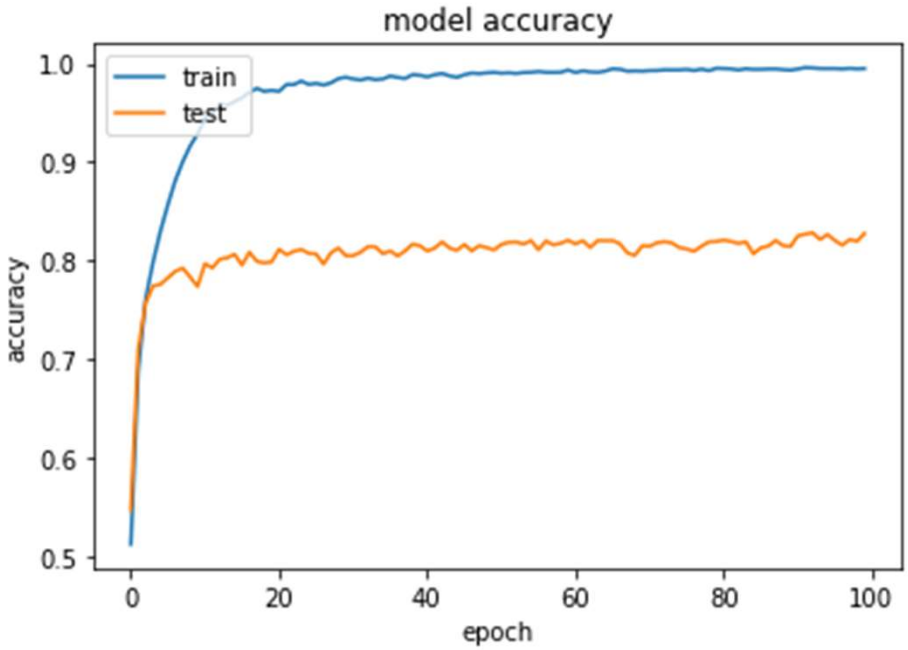
- Data augmentation is used to artificially increase the size of a training dataset by applying various transformations (flipping, rotation, scaling, cropping, shearing, and adding noise) to the existing data.

- It helps the model to learn invariant features and improves its ability to handle different variations and deformations

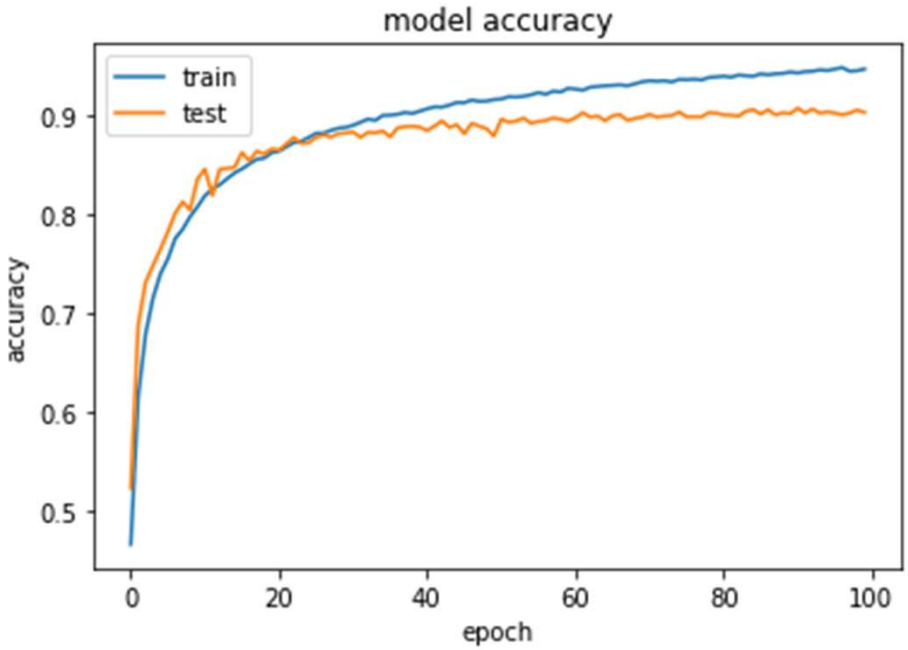
Deep Image [Wu et al. 2015]

http://arxiv.org/pdf/1501.02876v2.pdf

Illustration [With CIFAR-10 Data]



Without Image Augmentation. Clearly demonstrates overfitting



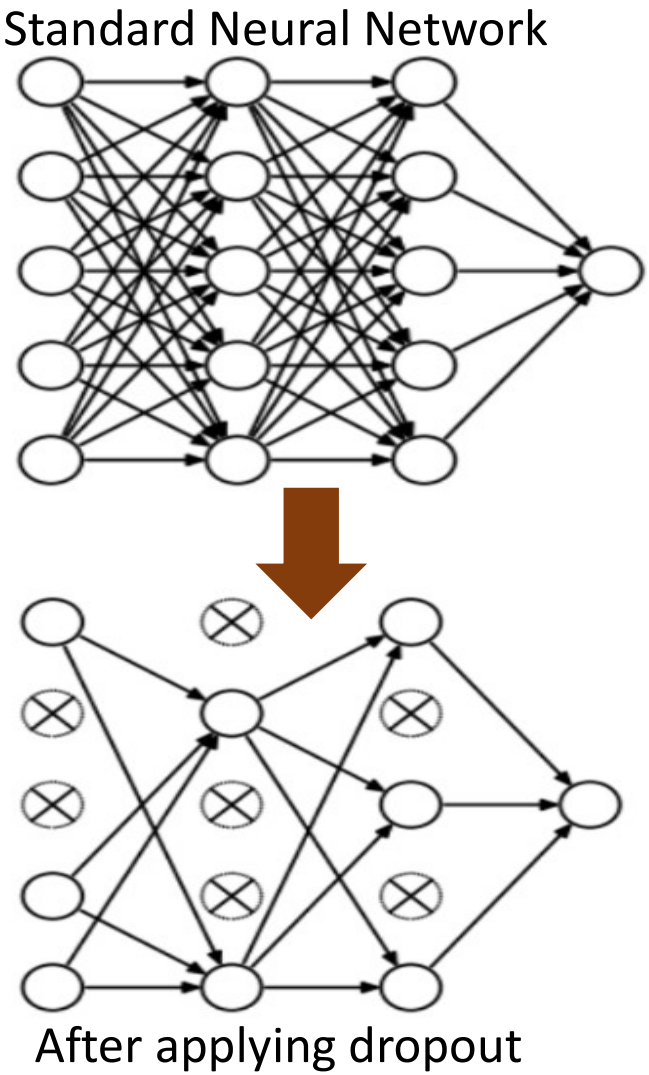
With Image augmentation (rotation, horizontal flip, shifts): 90.3% test accuracy, still rising after 100 epochs (ca. 2h training)

Dropout

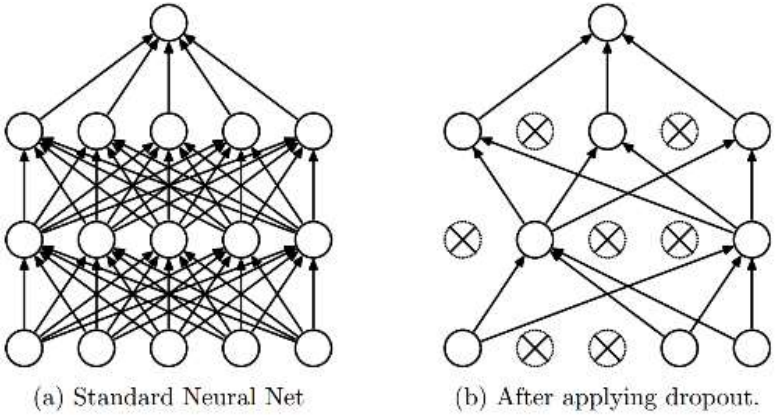
Intuition: successful conspiracies

- 50 people planning a conspiracy
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

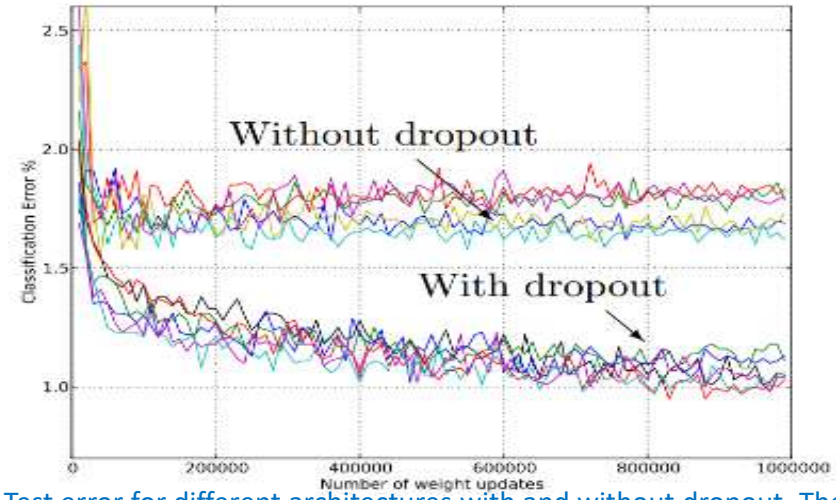
Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf



Dropout

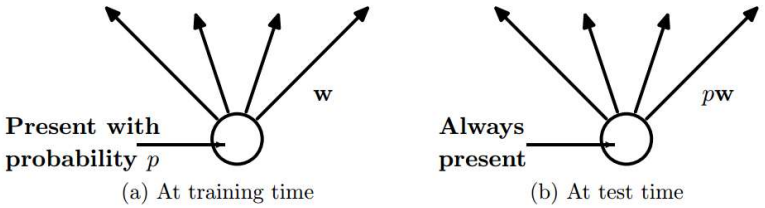


(a) Standard Neural Net (b) After applying dropout.



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Main Idea: approximately combining exponentially many different neural network architectures efficiently



(a) At training time (b) At test time

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Batch Normalization

- Batch Normalization (BN) is a normalization method/layer for neural networks.
- Usually inputs to neural networks are normalized
- A new layer is added so the gradient can "see" the normalization and made adjustments if needed.
  - The new layer has the power to learn the identity function to de-normalize the features if necessary!

https://arxiv.org/abs/1502.03167

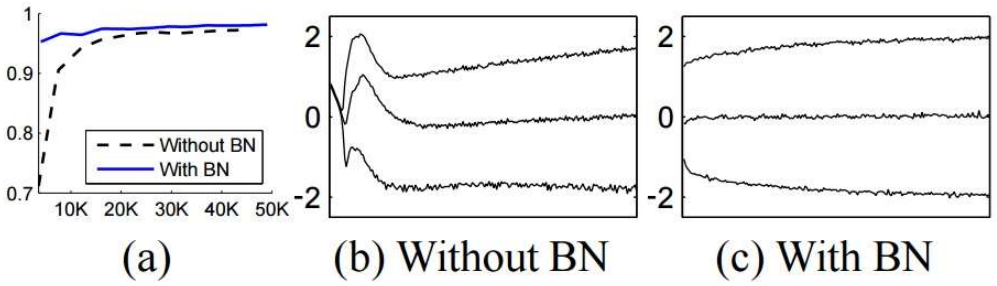
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

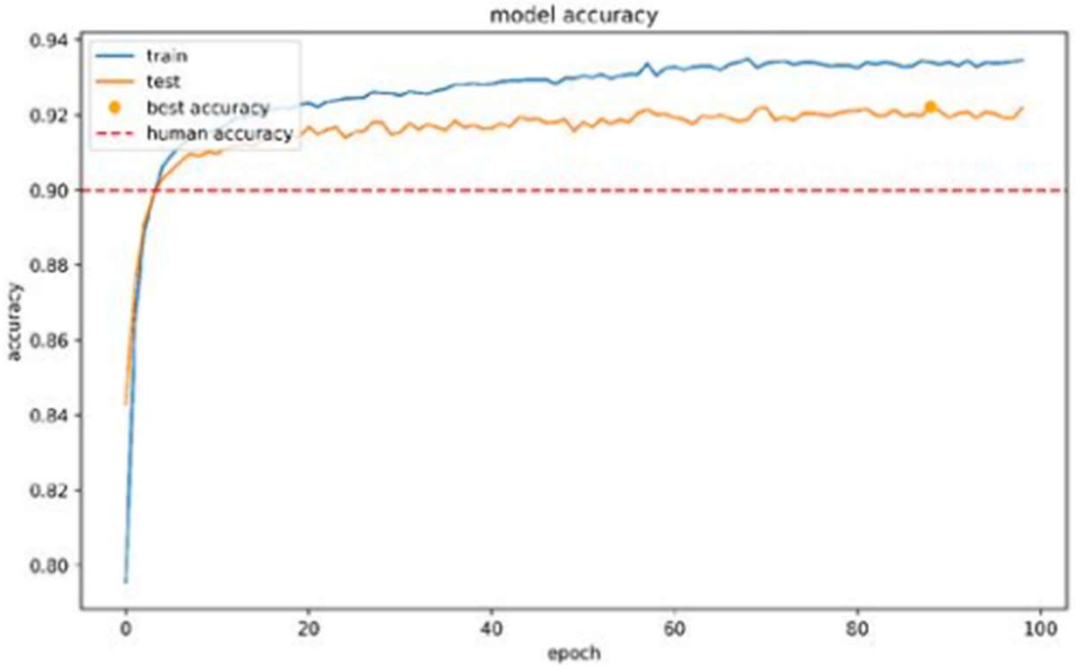
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

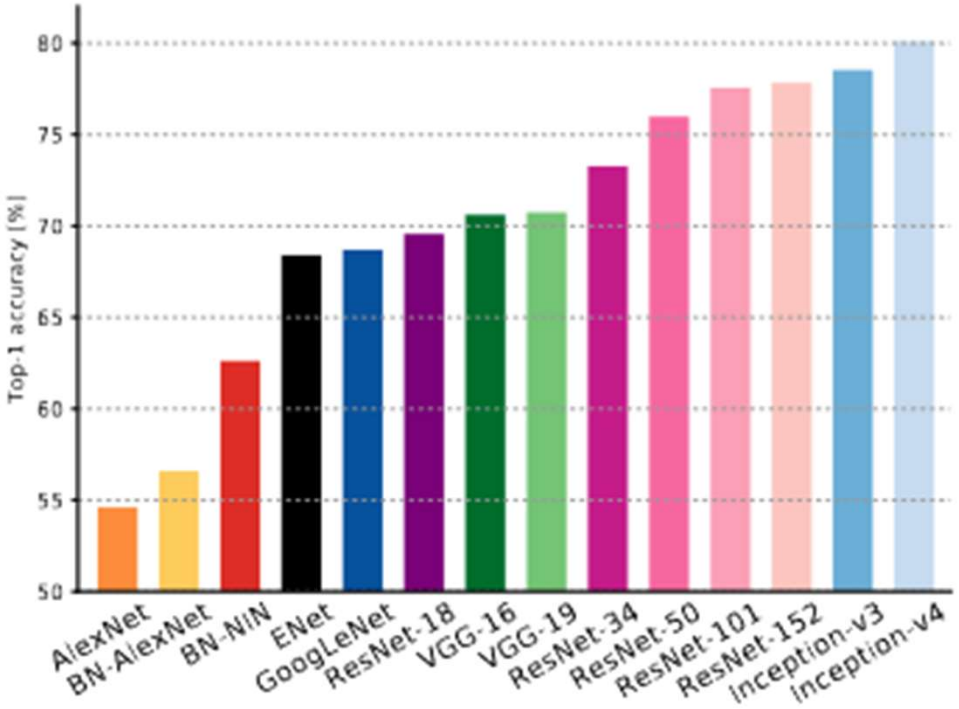


Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Perfect Learning Curve



Analysis of Pre-trained Networks



https://arxiv.org/pdf/1605.07678.pdf

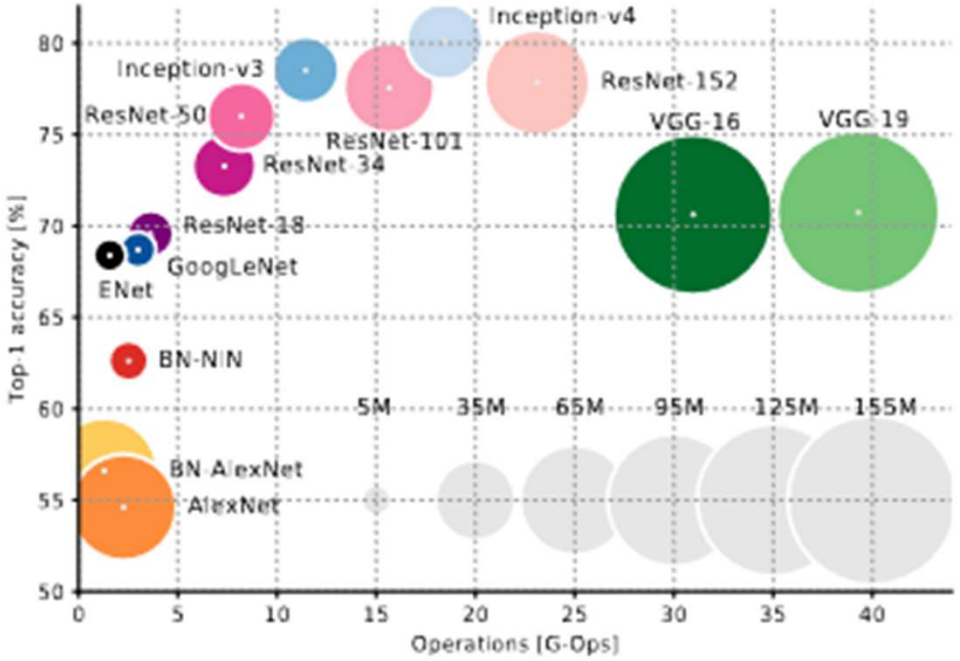
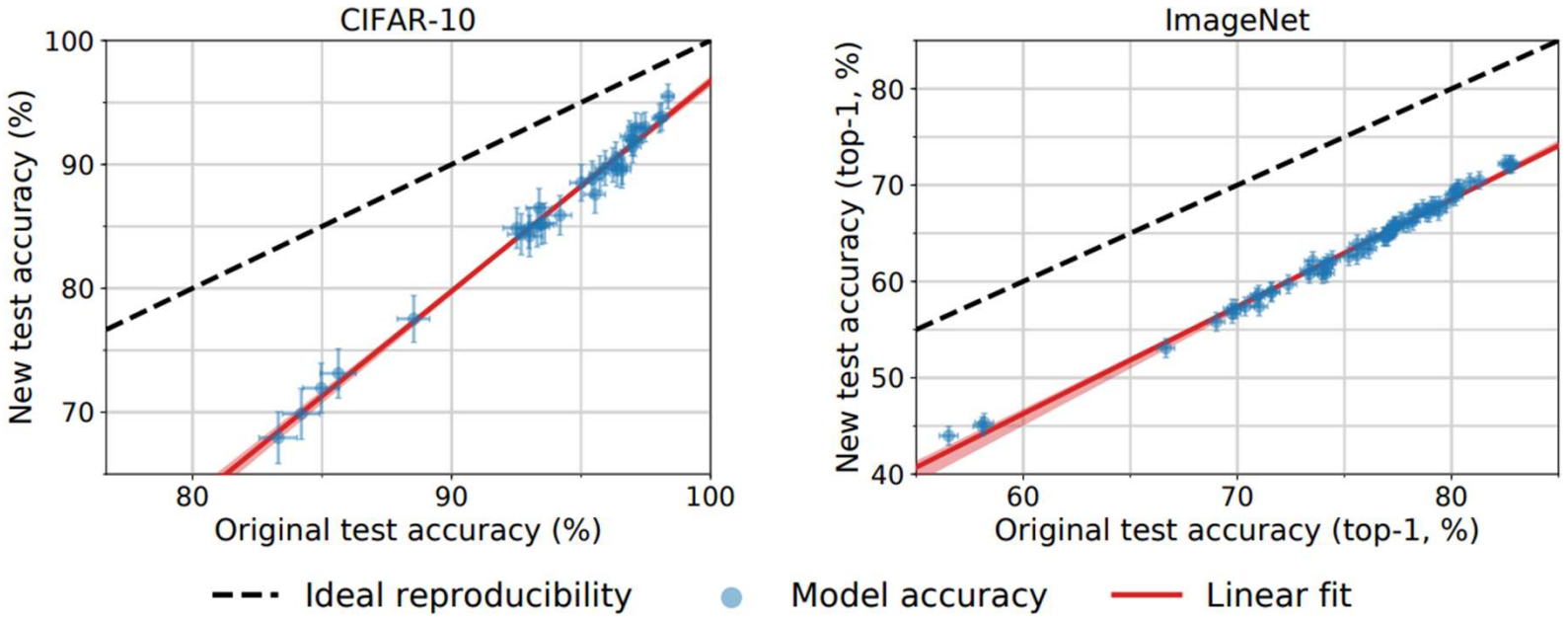


Image Source: arXiv:1605.07678

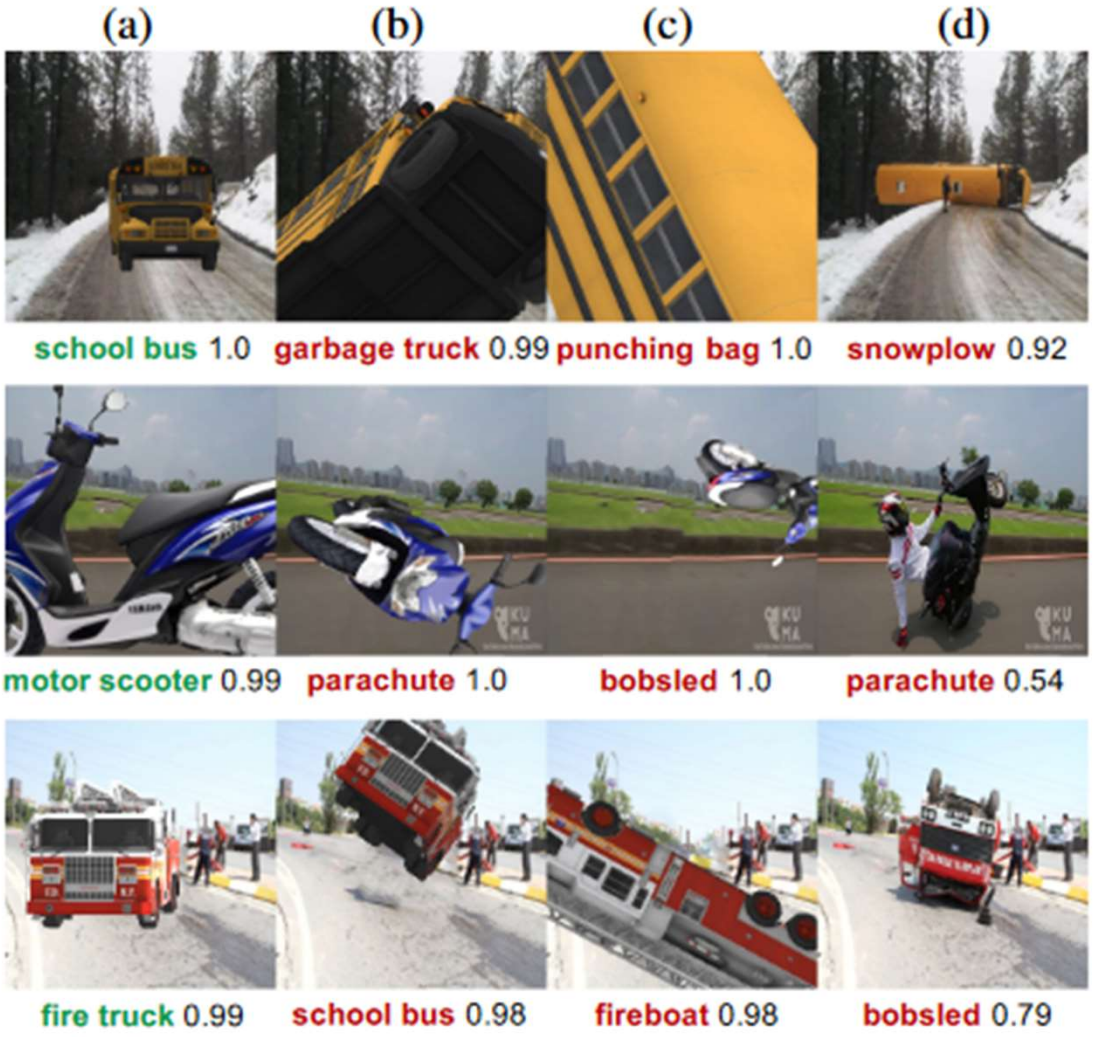
But deep networks are brittle



Do imagenet classifiers generalize to imagenet? (link)

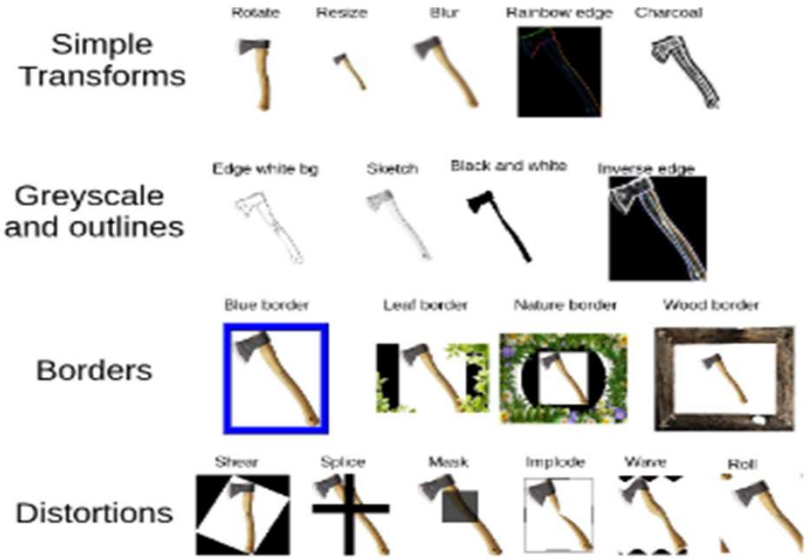
http://proceedings.mlr.press/v97/recht19a.html

… and fickle



https://arxiv.org/pdf/1811.11553.pdf

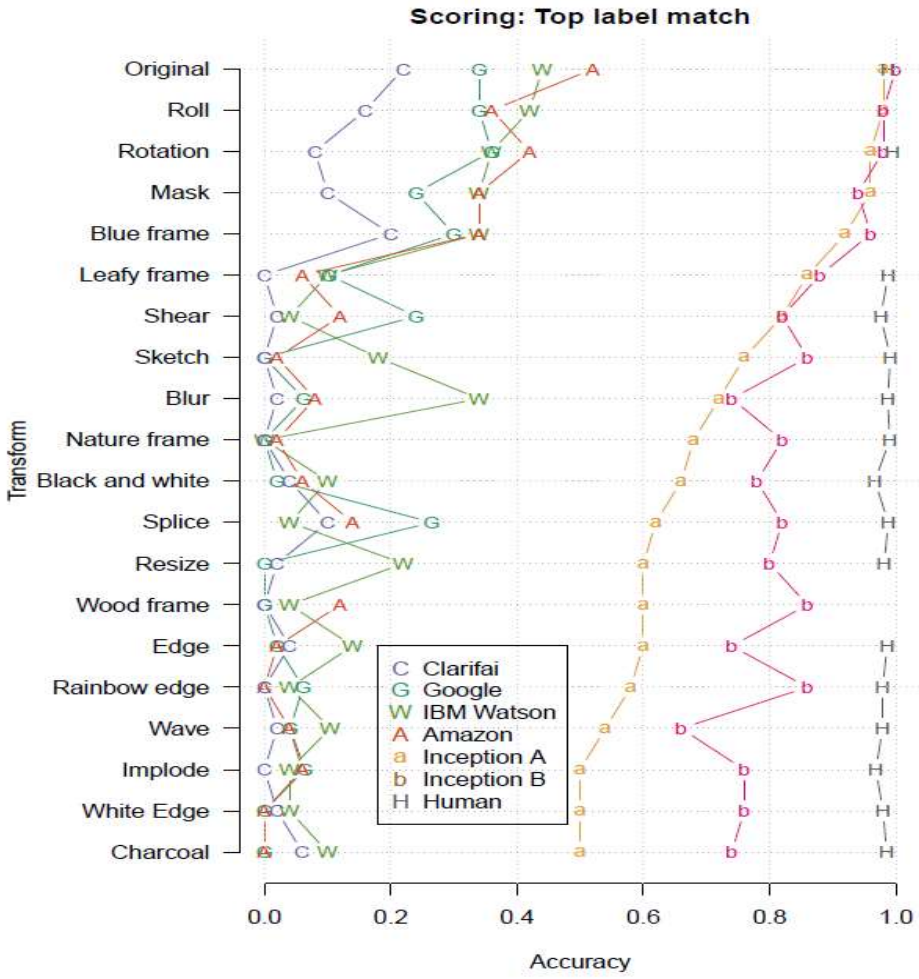
… and stupid



From the paper: The Unreasonable Ineptitude of Deep Image Classification Networks

… beyond belief

- Untransformed images are classified with 98% and 100% accuracy
- Transformed image accuracy drops enormously
- Human performance is unaffected
- Humans know when they are going to have trouble



From the paper: The Unreasonable Ineptitude of Deep Image Classification Networks

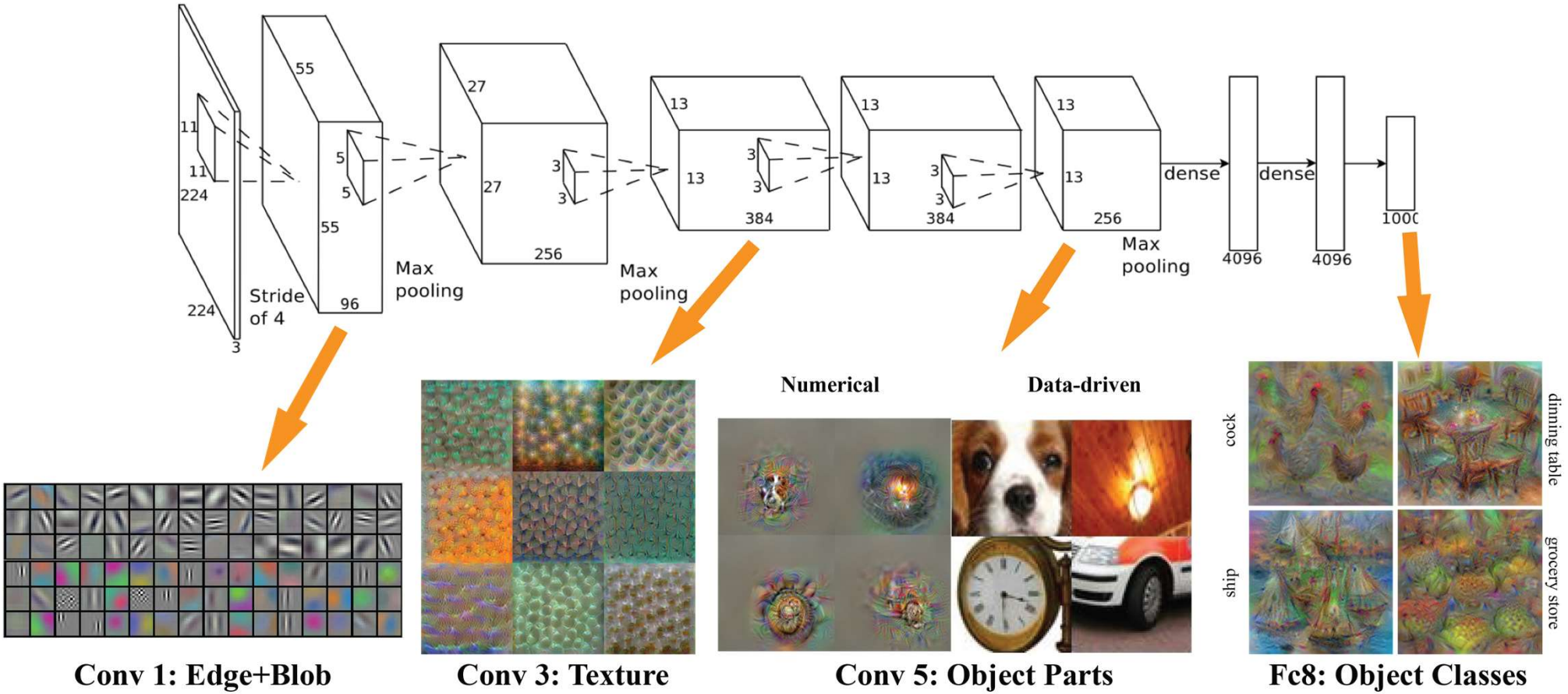
- Deep learning is based on large, high-quality datasets. What to do when you don’t have a large dataset?

- Lost in the wilderness of neural network’s hyperparameters with no idea where to start and what to configure????

Transfer Learning

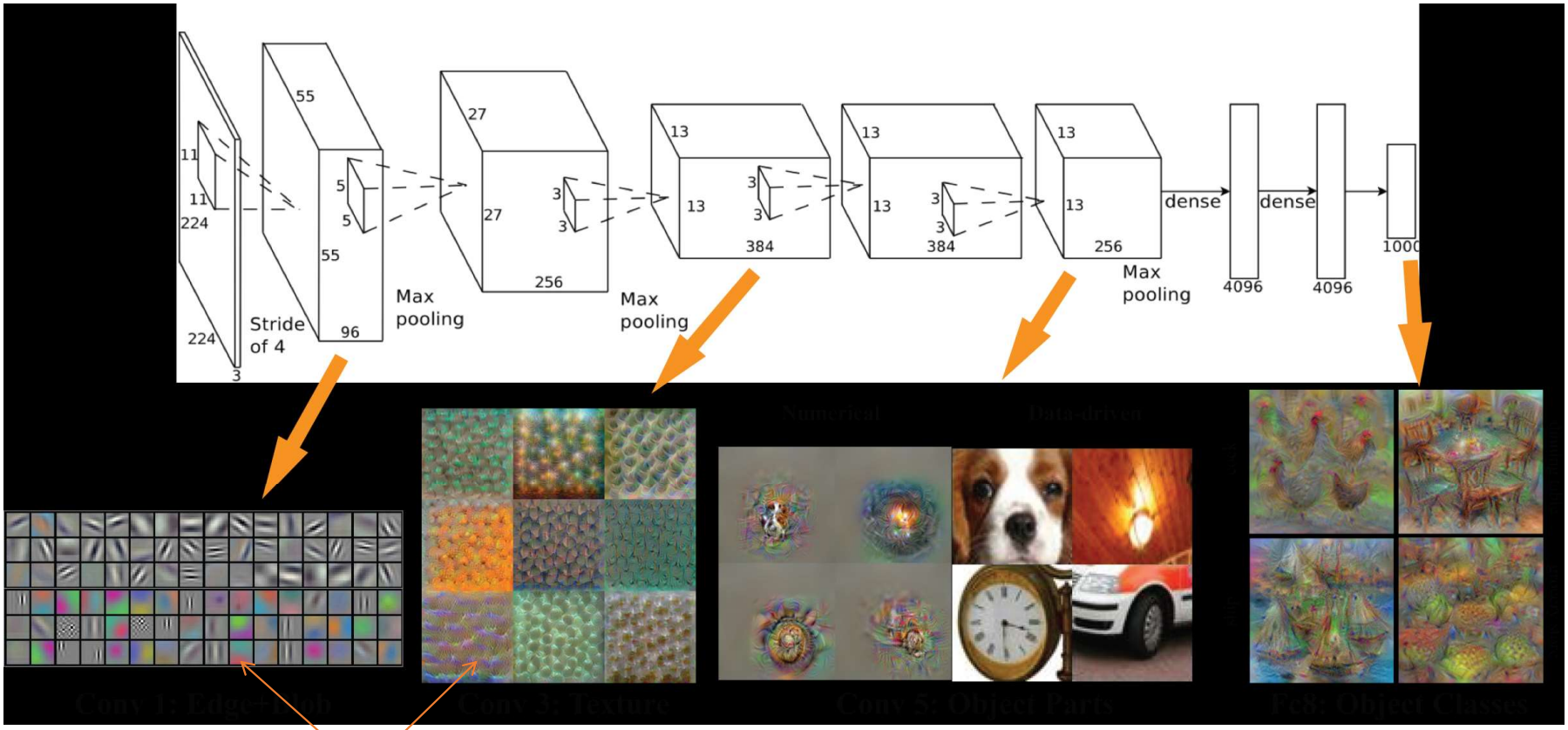
Fine-tuning or Freezing

Filters learnt by AlexNet



https://www.cc.gatech.edu/~hays/compvision/proj6/deepNetVis.png

Filters learnt by AlexNet

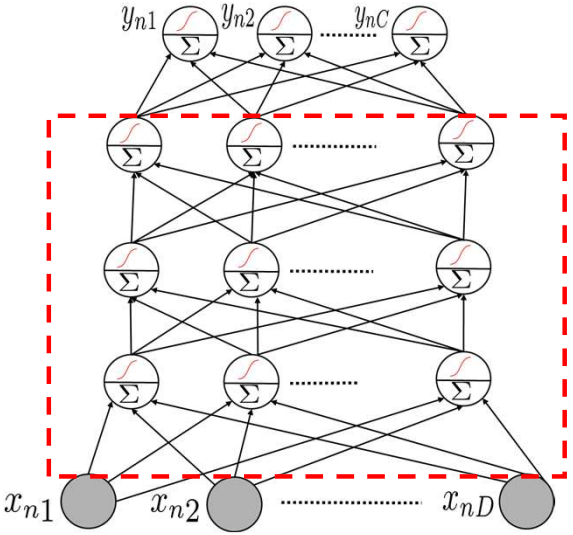


These features seem generic enough. Can we re-use them ?

https://www.cc.gatech.edu/~hays/compvision/proj6/deepNetVis.png

Using a Pre-trained Network

- A deep NN already trained in some "generic" data can be useful for other tasks, e.g.,
  - **Feature extraction**: Use a pre-trained net, remove the output layer, and use the rest of the network as a feature extractor for a related dataset



This part of a pre-trained net can be used as a feature extractor on some new task

Many packages, like Tensorflow and PyTorch provide such pre-trained module ready to be used

Sometimes also known as "transfer learning" in the context of neural nets

  - **Fine-tuning**: Use a pre-trained net, use its weights as initialization to train a deep net for a new but related task (useful when we don't have much training data for the new task)

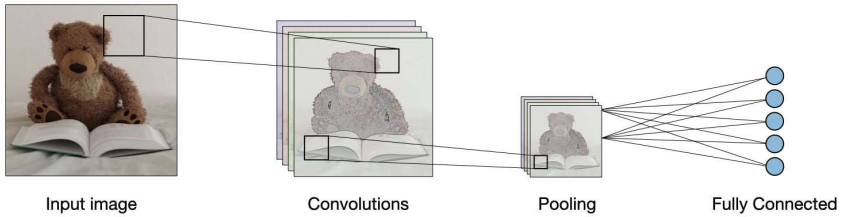
Transfer Learning

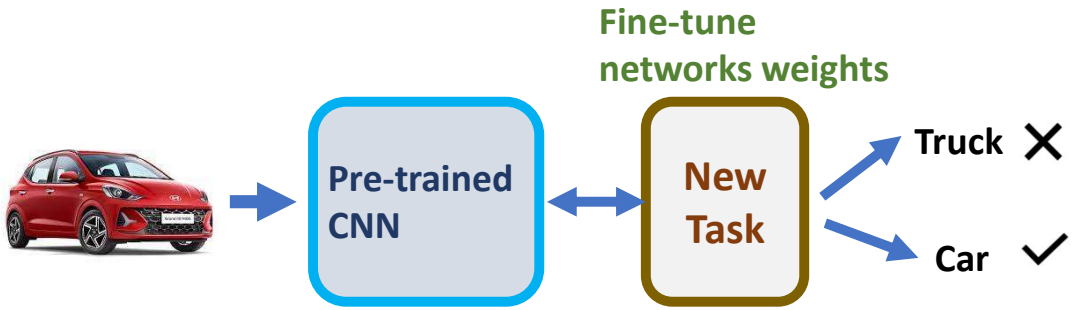
- Improvement of learning in a **new** task through the *transfer of knowledge* from a **related** task that has already been learned.

- No need to train from scratch – Leverage past powerful deep learning models and transfer their knowledge to the specific domain

- Two major strategies:
  - ConvNet as fixed feature extractor
  - Fine-tuning the ConvNet

CNN vs Transfer Learning

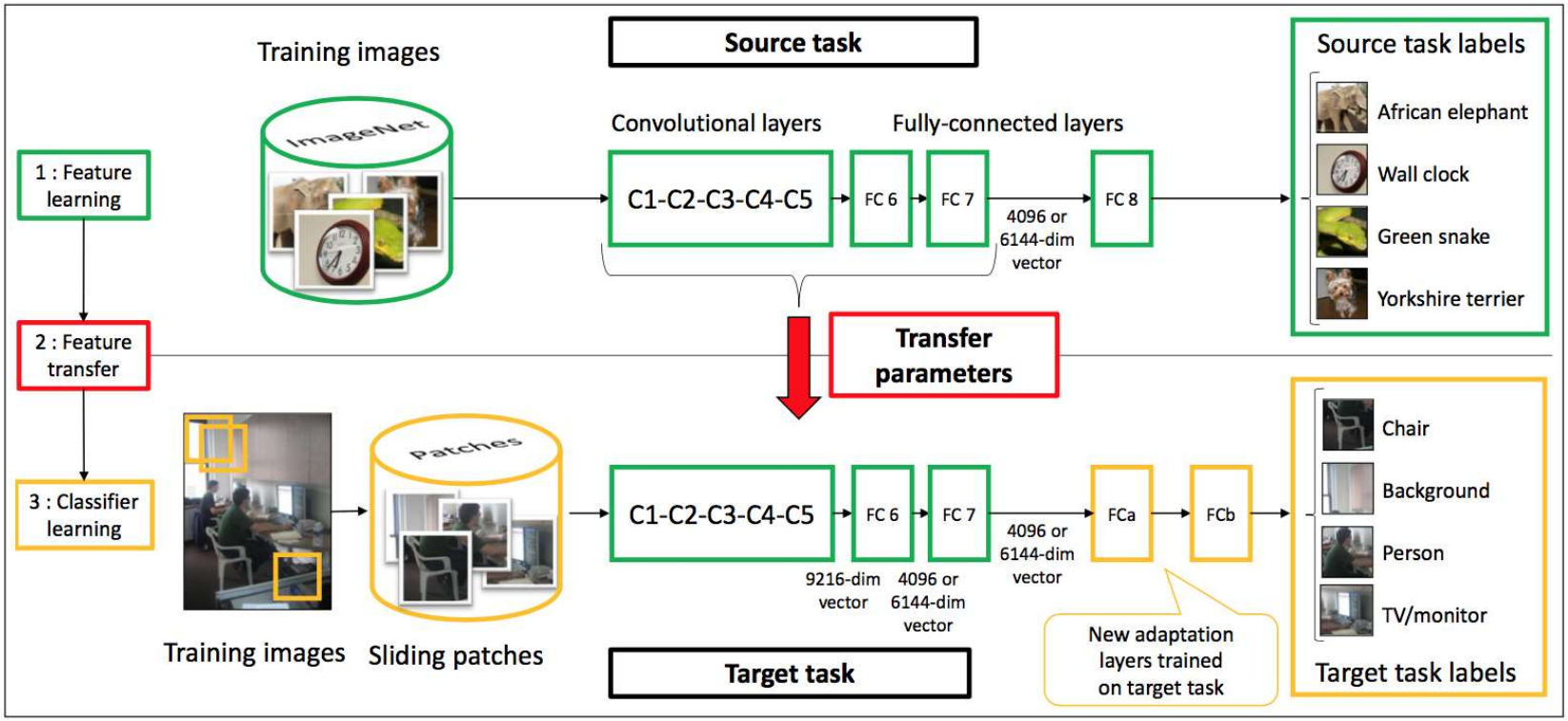




Training Data	1000s to millions of labelled images
Computation	Compute intensive
Training Time	Days to Weeks for real problems
Model Accuracy	High (can be overfitting to small datasets)

Training Data	100s to 1000s of labelled images
Computation	Moderate Computation
Training Time	Seconds to minutes
Model Accuracy	Good, depends on the pre-trained CNN model

Transfer Learning



Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks [Oquab et al. CVPR 2014] https://www.cv-foundation.org/openaccess/content_cvpr_2014/html/Oquab_Learning_and_Transferring_2014_CVPR_paper.html

Transfer Learning Details

- Prior knowledge from one domain and task can be applied to another domain and task
- $\mathcal{D} = \{\mathcal{X}, p(\mathcal{X})\}$, where $\mathcal{X}$ represents the feature space, with feature space including the input parameters
- A task in a domain, $\mathcal{T} = \{\mathcal{Y}, f(.)\}$, where $\mathcal{Y}$ represents the label space and $f(.)$ is the predictive function that relates the features and the labels
- Given a source domain $\mathcal{D}_S$ with a corresponding task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ with a task $\mathcal{T}_T$, the goal is to improve the target predictive function $f_T(.)$ by using knowledge learned from $\mathcal{D}_S$ and $\mathcal{T}_S$. Here $\mathcal{D}_S \neq \mathcal{D}_T$ and/or $\mathcal{T}_S \neq \mathcal{T}_T$
- $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$ becomes the case of traditional ML

Transfer Learning Approaches

Learning Strategy	Related Areas	Source and Target Domains	Source Domain Labels	Target Domain Labels	Source and Target Tasks	Tasks
Inductive	Multi-task learning	Same	Available	Available	Different but Related	Regression, Classification
	Self-taught Learning		Unavailable			
Transductive	Domain Adaptation, Co-variate shift	Different but related	Available	Unavailable	Same	Regression, Classification
Unsupervised		Different but related	Unavailable	Unavailable	Different but Related	Clustering, Dimensionality Reduction

Transfer Learning Strategies

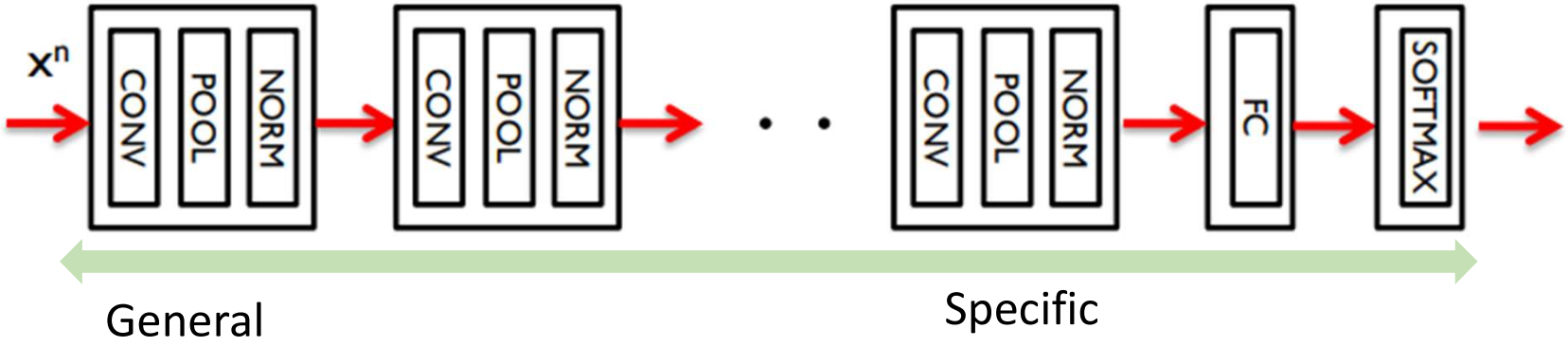
- ***Which*** part of the knowledge can be transferred from the source to the target to improve the performance of the target task?
- ***When*** to transfer and when not to, so that one improves the target task performance/results and does not degrade them?
- ***How*** to transfer the knowledge gained from the source model based on our current domain/task?

When and how to fine-tune?

	You have little data	You have a lot of data
Datasets are similar	**Train a classifier (Linear SVM) on top of bottleneck features**	**Fine-tune several or all layers**
Datasets are different	**Train a classifier on deep features of the CNN**	**Fine-tune all layers (use pre-trained weights as an initialization for your CNN)**

How transferable are features in CNN?

- A key observation that we noticed in visualization:-



- Further Questions?
  - Can we quantify the degree to which a particular layer is general or specific?
  - Does the transition occur suddenly at a single layer, or is it spread out over several layers?
  - Where does this transition take place: near the first, middle, or last layer of the network?

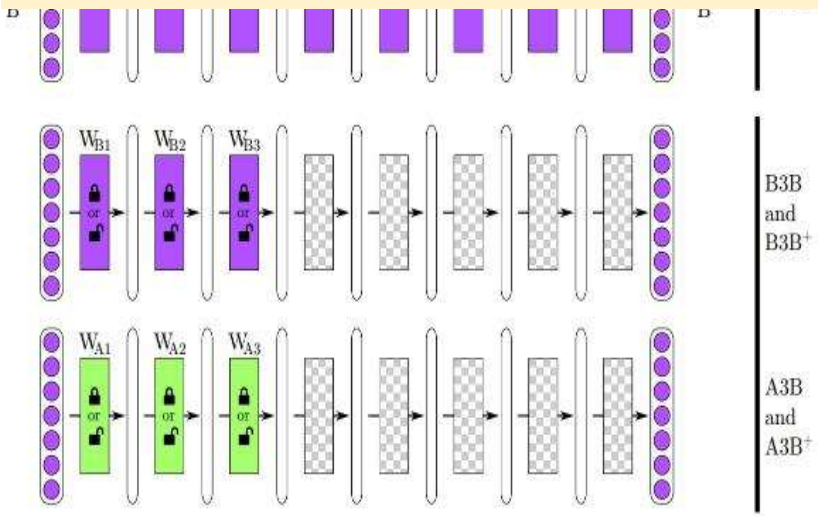
https://arxiv.org/abs/1411.1792

[Yosinski NIPS 2014]

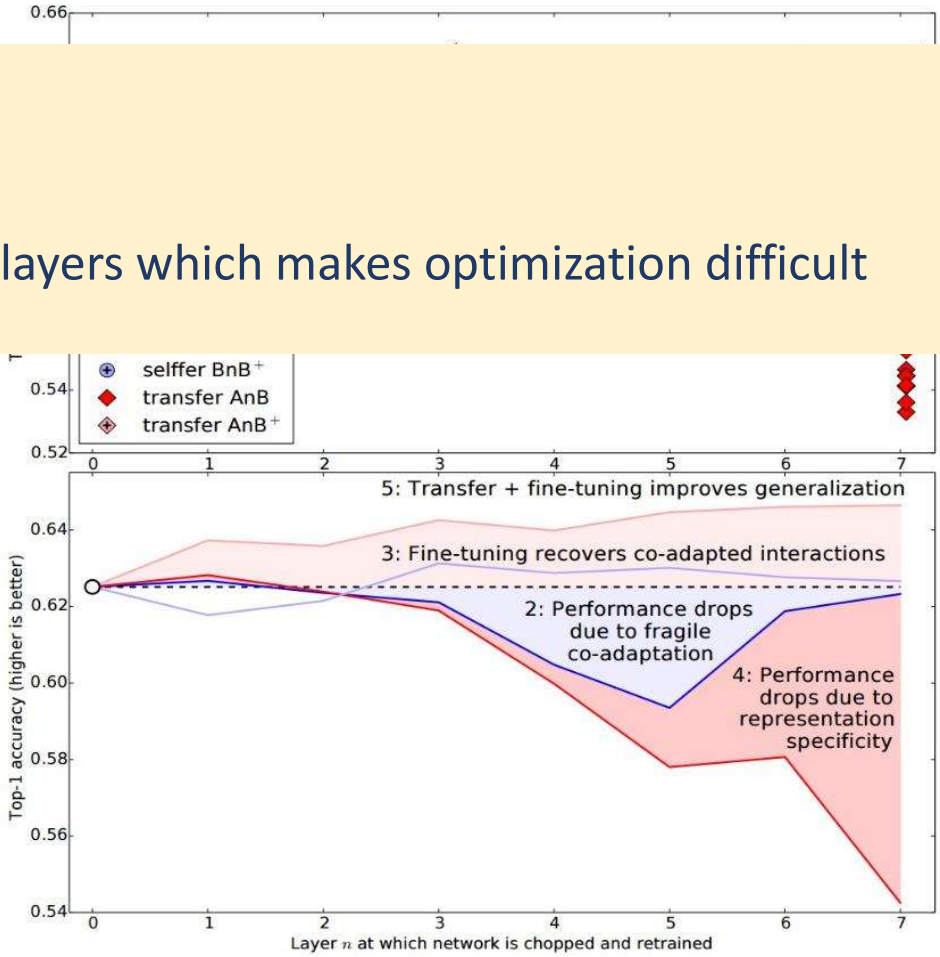
How transferable are features in CNN?

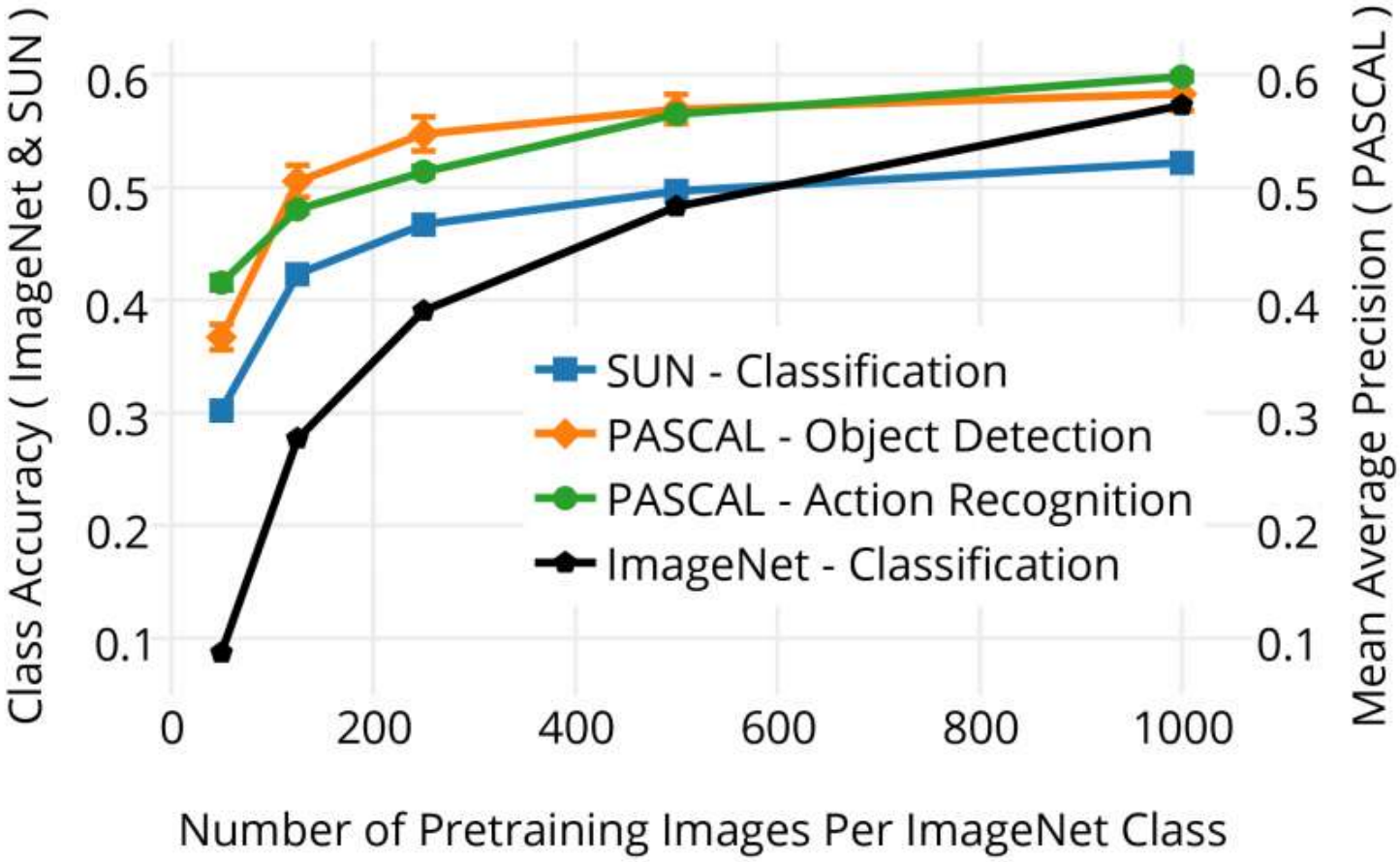
Observations

- Higher level neurons are more specialized
- There exists co-adapted neurons between layers which makes optimization difficult



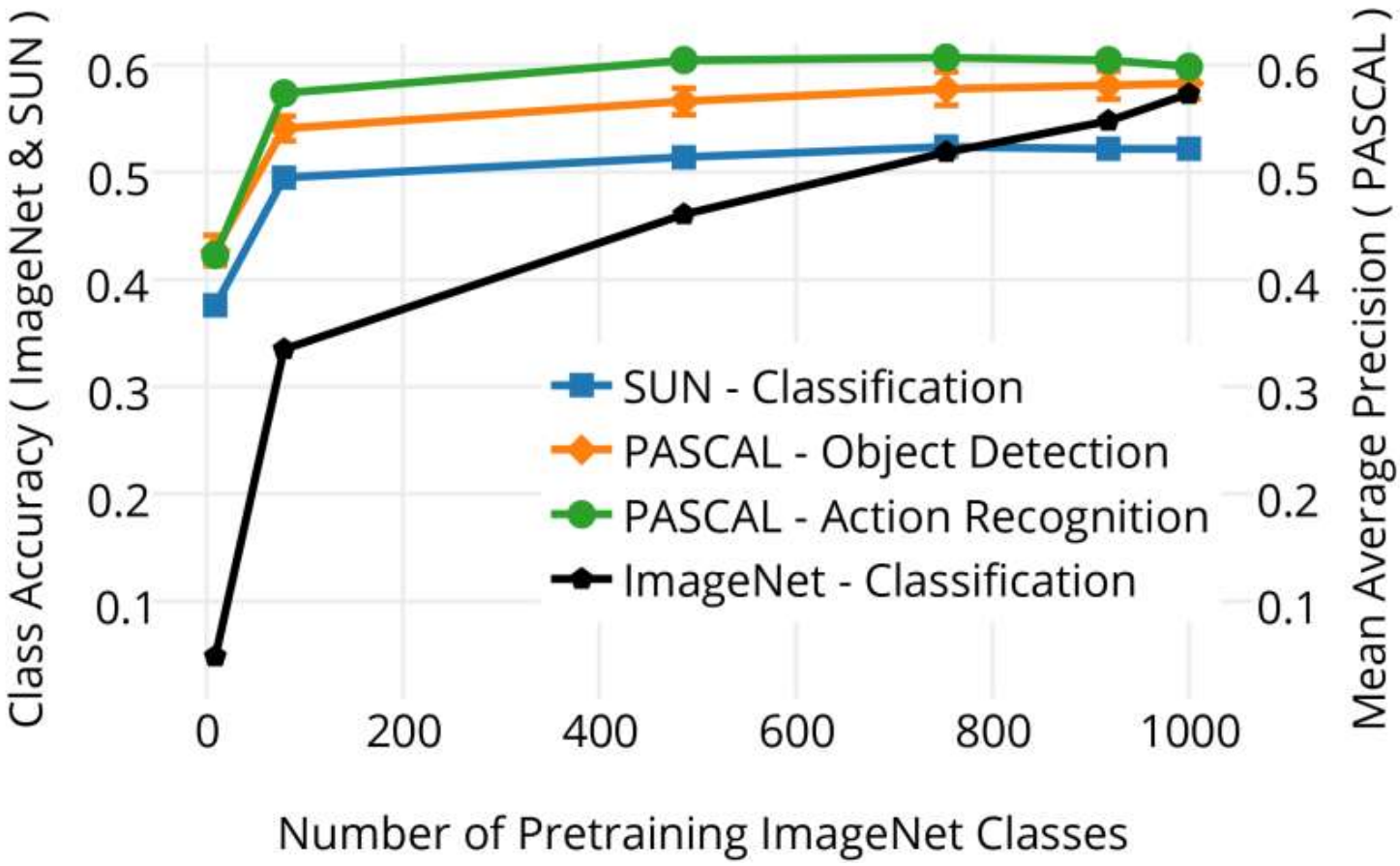
How transferable are features in deep neural networks [Yosinski NIPS 2014]



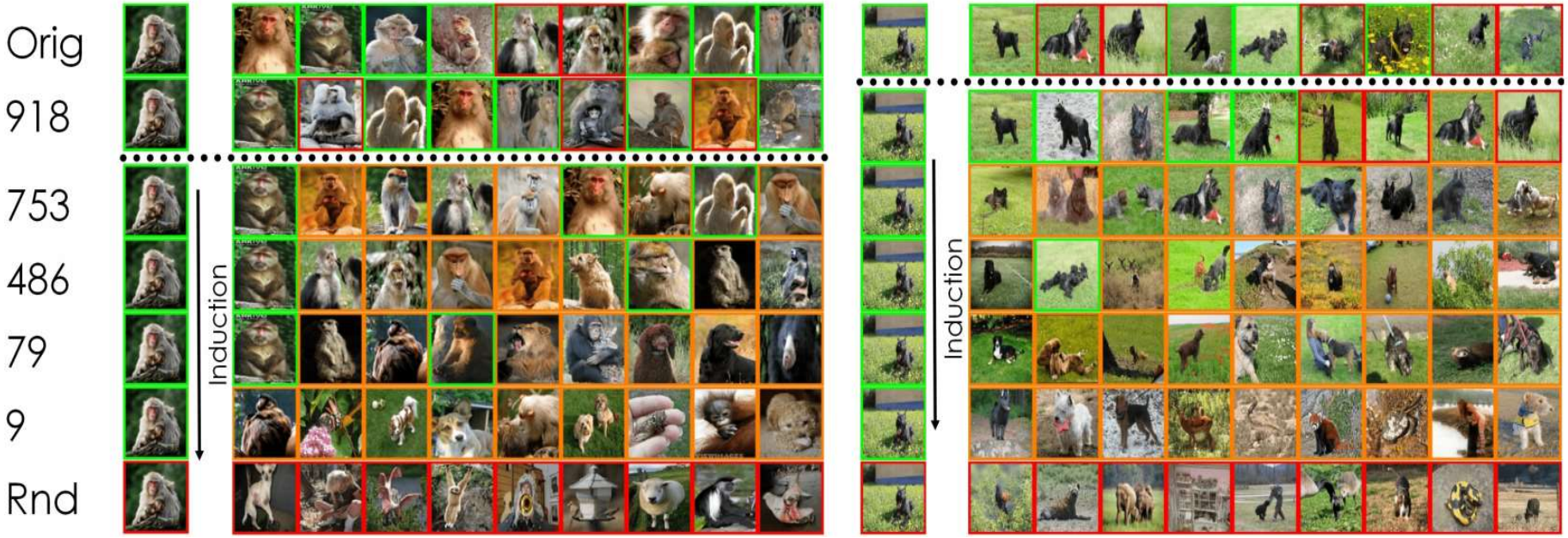


What makes ImageNet good for transfer learning? [Huh arXiv 2016]

https://arxiv.org/abs/1608.08614



What makes ImageNet good for transfer learning? [Huh arXiv 2016]



What makes ImageNet good for transfer learning? [Huh arXiv 2016]

Keras Application for Pre-trained Model

Learning with Little Data

- Setup: Only 2000 training examples (1000 per class). Additional 400 samples per class as validation data.

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Reference: Keras Blog

Learning with Little Data

- Data augmentation: “Augment" the training data via a number of random transformations. This helps the model generalize better.

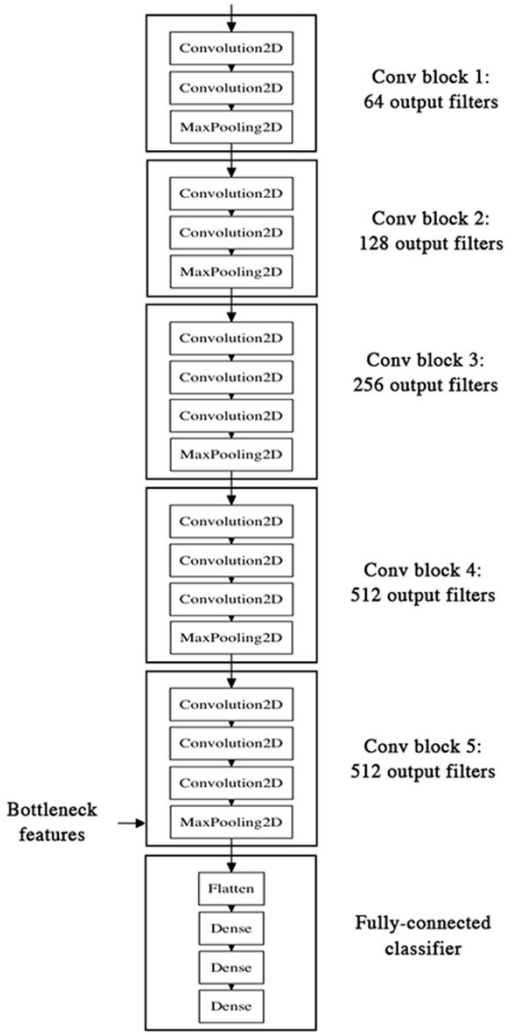
- Train a small convnet from scratch.

Reference: Keras Blog

Learning with Little Data

- Using the bottleneck features of a pre-trained network. No need for data augmentation
- A validation accuracy of 90%-91% is reached

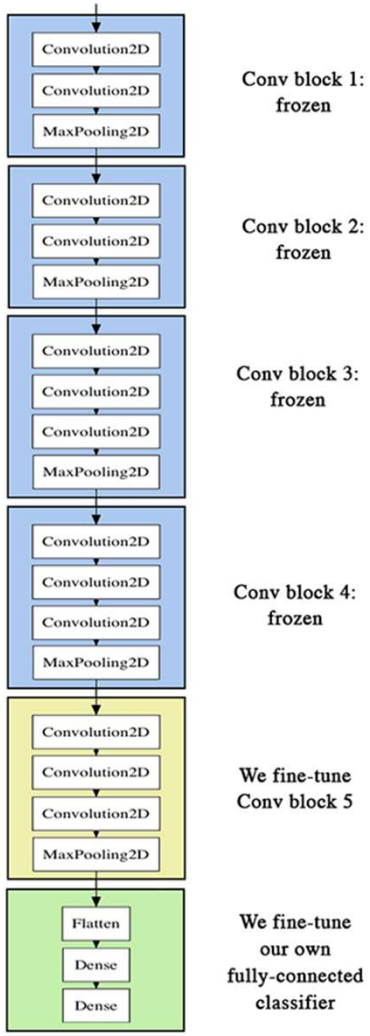
```
Train on 2000 samples, validate on 800 samples
Epoch 1/50
2000/2000 [==============================] - 1s - loss: 0.8932 - acc: 0.7345 - val_loss: 0.2664 - val_acc: 0.8862
Epoch 2/50
2000/2000 [==============================] - 1s - loss: 0.3556 - acc: 0.8460 - val_loss: 0.4704 - val_acc: 0.7725
...
Epoch 47/50
2000/2000 [==============================] - 1s - loss: 0.0063 - acc: 0.9990 - val_loss: 0.8230 - val_acc: 0.9125
Epoch 48/50
2000/2000 [==============================] - 1s - loss: 0.0144 - acc: 0.9960 - val_loss: 0.8204 - val_acc: 0.9075
Epoch 49/50
2000/2000 [==============================] - 1s - loss: 0.0102 - acc: 0.9960 - val_loss: 0.8334 - val_acc: 0.9038
Epoch 50/50
2000/2000 [==============================] - 1s - loss: 0.0040 - acc: 0.9985 - val_loss: 0.8556 - val_acc: 0.9075
```



Reference: Keras Blog

Learning with Little Data

- Fine-tuning the top layers of a pre-trained network
  - instantiate the convolutional base of VGG16 and load its weights
  - add our previously defined fully-connected model on top, and load its weights
  - freeze the layers of the VGG16 model up to the last convolutional block
- A validation accuracy of 94% is reached after 50 epochs



Reference: Keras Blog

Things to remember

- Training CNN – Prevention of Overfitting
  - Dropout
  - Data augmentation
  - Batch normalization
- Transfer learning
  - Two strategies
    - CNN code
    - Fine-tuning
  - When and how to transfer
  - Characteristics of transfer learning