

Applications: Large-Scale Deep Learning

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics in Applications

1. Large-Scale Deep Learning
2. Computer Vision
3. Speech Recognition
4. Natural Language Processing
5. Other Applications

Topics in Large Scale Deep Learning

- Large scale Deep Learning
 1. Fast CPU implementations
 2. GPU implementations
 3. Large-scale distributed implementations
 4. Model compression
 5. Dynamic structure
 6. Specialized hardware implementations

Need for Large Networks

- Large Scale Neural Networks
 - No of neurons is large
- Large Scale networks are needed for most serious AI applications
 - Dramatic increase in size is what improved from 1980s to today

Specialized Large Networks

- While deep learning is general, some specialization is required
 1. Vision tasks require a large no. of input features (pixels)
 2. Language tasks require large no. of possible values (words in vocabulary) per input feature

Examples of Large Networks

- Keras ships with five pretrained CNNs
 - Trained on ImageNet dataset (22,000 object categories):
 1. VGG16
 2. VGG19
 3. ResNet50
 4. Inception V3
 5. Xception

```
VGGNet, ResNet, Inception, and Xception with Keras
1. # import the necessary packages
2. from keras.applications import ResNet50
3. from keras.applications import InceptionV3
4. from keras.applications import Xception # TensorFlow ONLY
5. from keras.applications import VGG16
6. from keras.applications import VGG19
7. from keras.applications import imagenet_utils
8. from keras.applications.inception_v3 import preprocess_input
9. from keras.preprocessing.image import img_to_array
10. from keras.preprocessing.image import load_img
11. import numpy as np
12. import argparse
13. import cv2
```

VGG16,19: Visual Geometry Group, Oxford

```
VGGNet, ResNet, Inception, and Xception with Keras
$ python classify_image.py --image images/soccer_ball.jpg --model vgg16

(keras2.0)malcolm:keras-networks adrianrosebrock$ python classify_image.py --image
[INFO] loading vgg16...
Label: soccer_ball, 93.43%

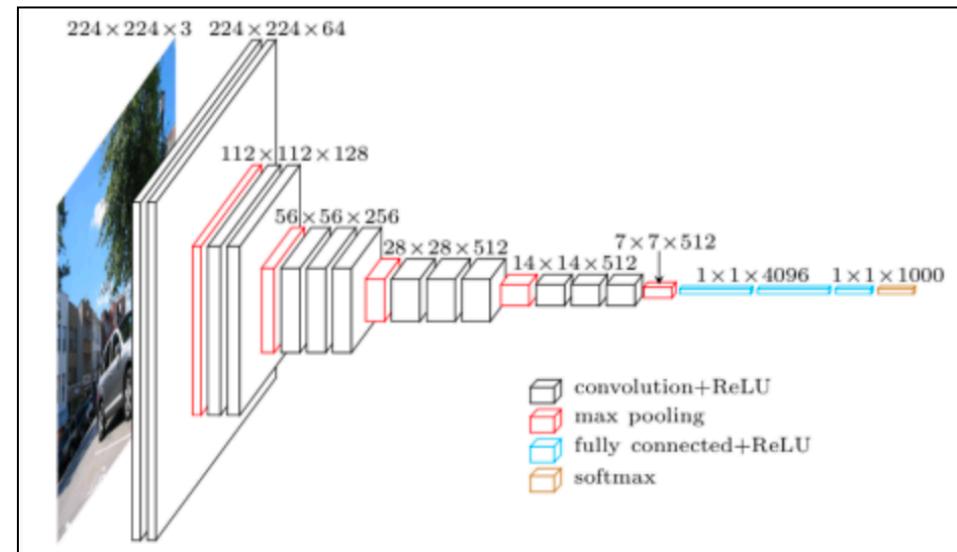
```

```
VGGNet, ResNet, Inception, and Xception with Keras
$ python classify_image.py --image images/bmw.png --model vgg19

(keras2.0)malcolm:keras-networks adrianrosebrock$ python classify_image.py --image
[INFO] loading vgg19...
Label: convertible, 91.76%

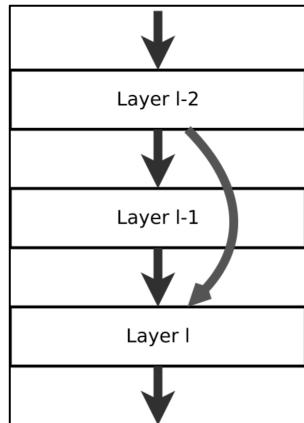
```

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
FC-4096					
FC-4096					
FC-1000					
soft-max					

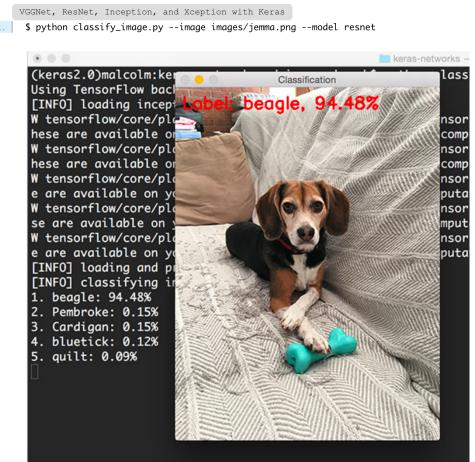


Large Networks: ResNet

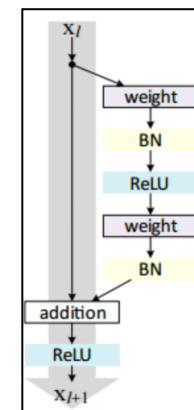
Canonical form of a residual neural network.
A layer $\ell - 1$ is skipped over activation from $\ell - 2$



Motivation:
avoid vanishing gradients,
by reusing activations from a
previous layer until the
adjacent layer learns its weights.

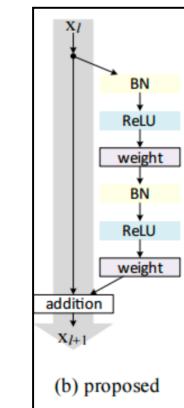


Residual module
in ResNet as originally
proposed



Demonstrated that
extremely deep networks
can be trained using standard SGD

Updated Resnet
using pre-activation.



Large Networks: Inception Net

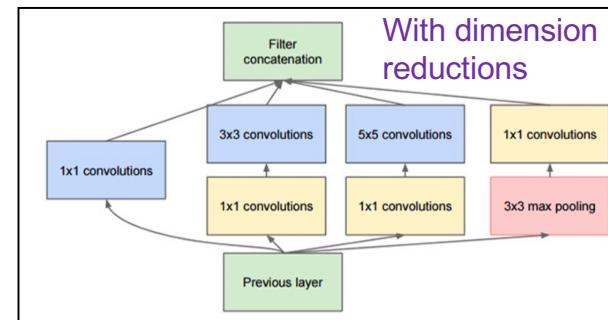
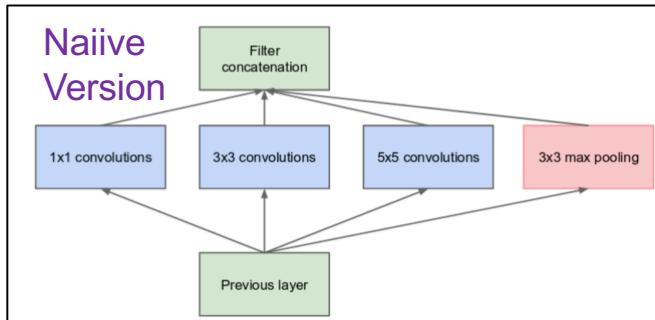
Uses lots of tricks to push performance

With wide variation in location of information,
choice of kernel size tough.

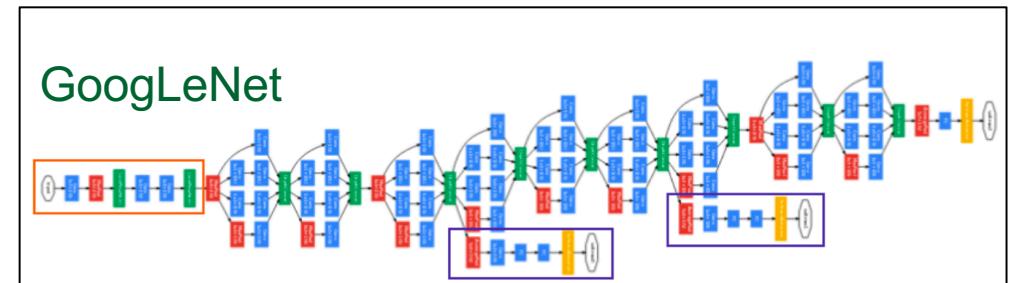
Larger kernel for information that is distributed more globally
Smaller kernel for information that is distributed more locally



An Inception Net uses filters with multiple sizes operating on the same level



GoogLeNet has 9 inception modules stacked linearly.
22 layers deep (27 with pooling layers).



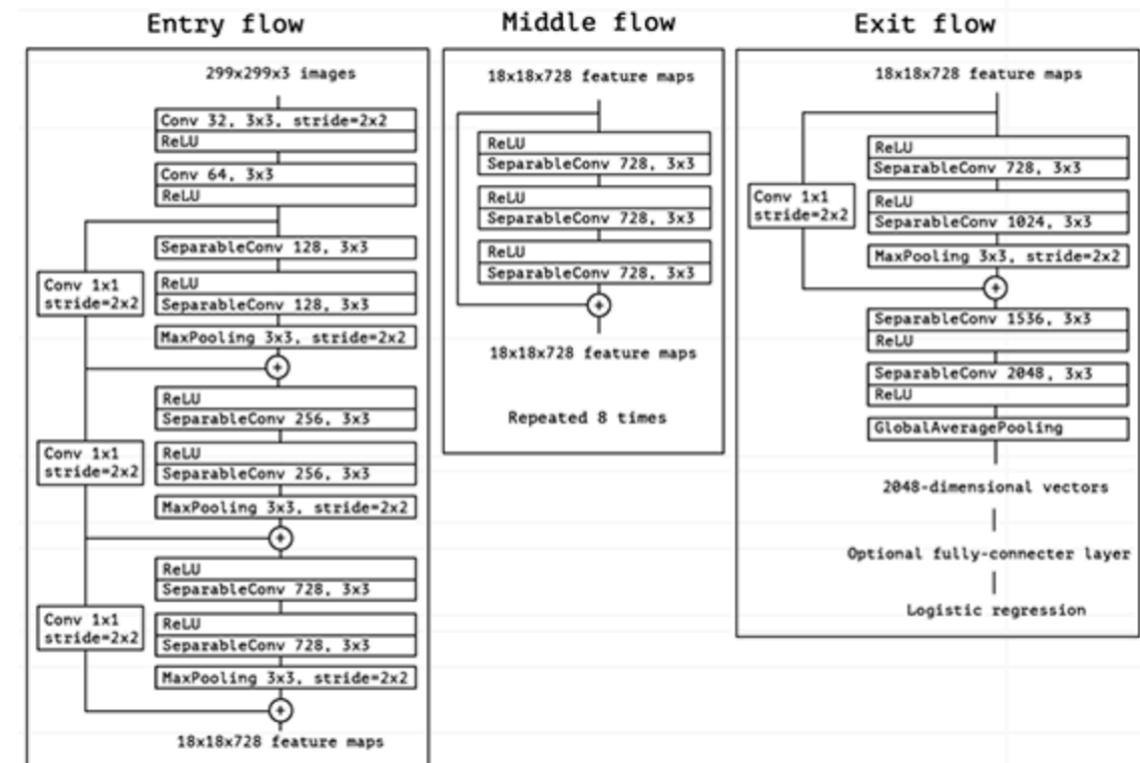
Global average pooling at end of last inception module.
As a very deep network, it is subject to **vanishing gradients**
To prevent the **middle part** of the network from “dying out”, **auxiliary classifiers** (Purple boxes)

Large Networks: Xception Net

Xception is an extension of the Inception architecture

Replaces standard Inception modules with depthwise separable convolutions.

It is 71 layers deep
and can classify
images into 1000 classes,



Large Scale Deep Learning

- Philosophy of connectionism
 - While an individual neuron/feature is not intelligent, a large no. can exhibit intelligent behavior
 - No. of neurons must be *large*, e.g. 10 **layers**, 100 **neurons** per layer
- Although network sizes have increased exponentially in three decades, ANNs are only as large as nervous systems of insects
- Since size is important, DL requires high-performance hardware and software infrastructure

Fast CPU implementations

- Traditionally neural nets were trained on CPU of a single machine, with careful design
 - In the past, trade-off between tuned fixed-point and floating point representations
 - Specialized numerical computation yielded high payoffs
- Today they are considered inadequate
 - Mostly use GPUs or networked CPUs

GPU Implementations

- Modern neural network implementations are based on GPUs
 - GPUs are specialized hardware components
 - Originally developed for graphics
 - Consumer market for video game rendering
 - High degree of parallelism, high bandwidth
- Performance characteristics for video gaming systems are beneficial for neural networks as well

GPU vs CPU

- GPU has many cores compared to CPU



- CPU: Intel Core i7 has 4 cores (operates at 3.5 GHz)
- GPU: NVIDIA GTX 690 has 3072 cores (500–1000 MHz)
 - Cuda cores are designed as SIMD
 - Perform a specific operation on a large data in parallel
 - Altering each pixel of an image, GPUs can do it faster
 - Performing matrix operations GPUs are the way to go

General Purpose GPUs

- General Purpose GPUs (GP-GPUs) can execute arbitrary code
 - Not just rendering subroutines
 - NVIDIA's CUDA programming language allows writing arbitrary code in a C-like language
- Convenient programming model, massive parallelism and high memory bandwidth make GP-GPUs the ideal platform for neural network programming

Writing Code for GPUs

- Writing efficient code for GP-GPUs is a difficult task best left to specialists
- Techniques for high performance code is very different for GPUs from CPUs
 - For CPUs efficient to read cached information
 - For GPUs it is faster to compute same value twice
- Done by models making calls to library of operations for convolution and matrix multiplication
- PyTorch and Tensorflow programs can run on CPU or GPU

Using GPUs from Pytorch

- **Sending net and code to GPU:**
 1. `device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")`
 2. `net = net.to(device)`
 3. `input = input.to(device)`
 4. `labels = labels.to(device)`
- **Code agnostic**
 - If GPU installed, then code runs on it else on CPU

Large Scale Distributed Implementations

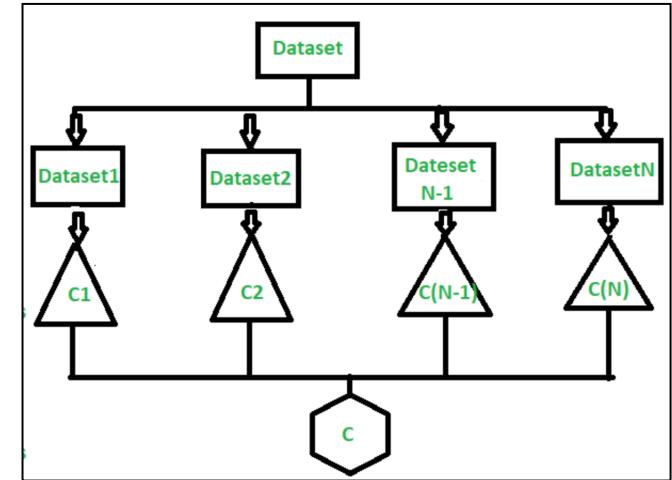
- When computational resources on a single machine are insufficient, distribute workload
- Distributing inference is simple
 - Each input example we want to process is run on a different machine
 - This is known as data parallelism
- Model parallelism
 - Multiple machines work on a single datapoint
 - Each machine runs on different part of model
 - Feasible for both training and inference

Model Compression

- Often time and memory for inference has to be low rather than time and memory for training
 - If personalization is not needed then we can train once and billions can use it
 - User more resource-constrained than developer
 - E.g., speech recognition trained on a cluster for use on mobile phones
- Strategy for reducing inference cost is *model compression*

Applicability of Model Compression

- When model size is large to prevent overfitting
 - Lowest generalization error is an ensemble of independently trained models



- Inference with many models is expensive
- A single model generalizes better if it is large
 - E.g., if it is regularized with dropout
 - These large models learn $f(\mathbf{x})$ but use many more parameters than needed

Reducing model size

- Large model size is needed due to limited training samples
- After we fit the function $f(\mathbf{x})$ we can generate infinitely many examples by applying f to randomly sampled points \mathbf{x}
- We then train a smaller mode to match $f(\mathbf{x})$ on these points
- Sample generation methods:
 1. Corrupt training points using noise
 2. Drawing samples from a generative model trained on original dataset

Dynamic Structure

- To accelerate data-processing systems is to build a *dynamic structure* in the graph describing the computation needed to process an input
- Dynamically determine which subset of many neural networks should be run on a given input
- Individual neural networks can determine which subset of features (hidden units) to compute given information from the input
- This also called *conditional computation*

Accelerating classifier inference

- A venerable strategy:
 - Use a Cascade of classifiers
 - Can be applied when a rare object is to be detected
- To know for sure that the object is present we need high capacity
- We can use much less computation to reject inputs as not containing the object
 - Train a sequence of classifiers
 - First classifiers have low capacity and trained to have high recall
 - Final classifier has high precision