# CS396: Selected CS2

## (Deep Learning for visual recognition)

**Spring 2022**

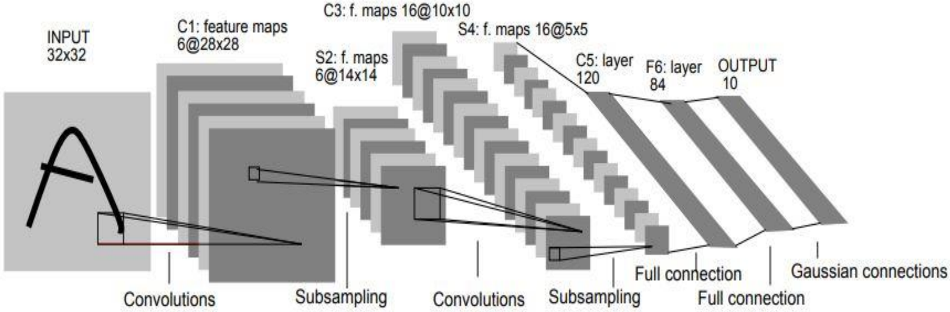
**Dr. Wessam EL-Behaidy**
Associate Professor, Computer Science Department,
Faculty of Computers and Artificial Intelligence,
Helwan University.

**Lectures (Course slides) are based on** Stanford course :
Convolutional Neural Networks for Visual Recognition (CS231n):
http://cs231n.stanford.edu/index.html

# Lecture 6: Convolution Neural Network (Part 2)

# Case Study: LeNet-5

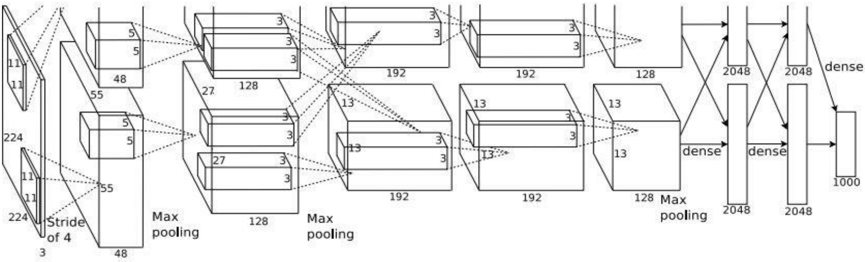
[LeCun et al., 1998]




Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*

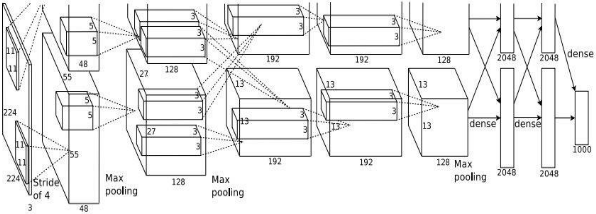



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

Trained on two GTX 580 GPUs for **five to six days**.

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*




Input: 227x227x3 images

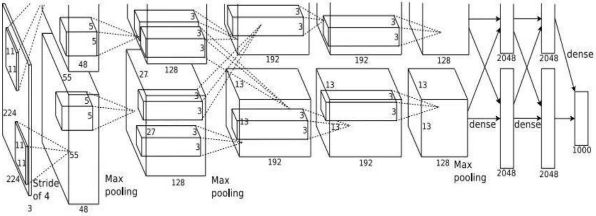
**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*



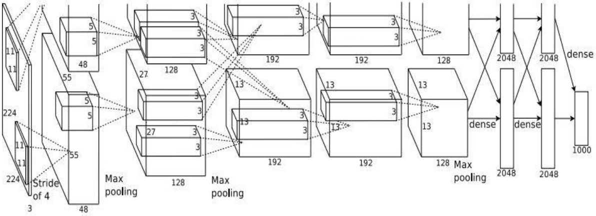

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*




Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

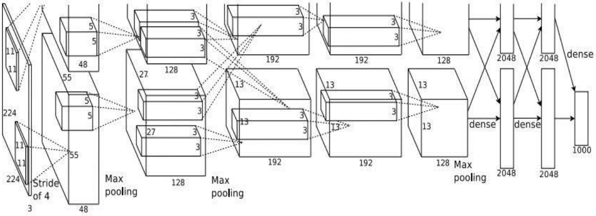
=>

Output volume **[55x55x96]**

Parameters: (11*11*3)*96 = **35K**

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*



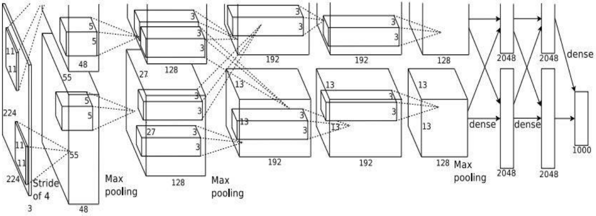

Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*



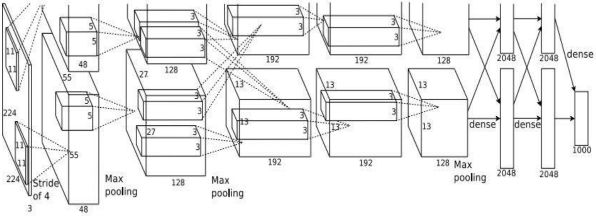

Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*

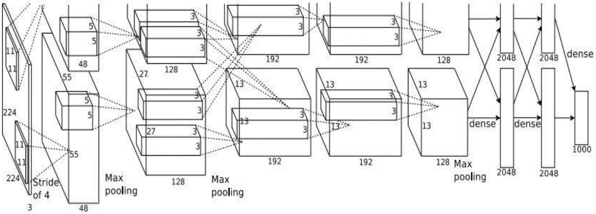



Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

# Case Study: AlexNet

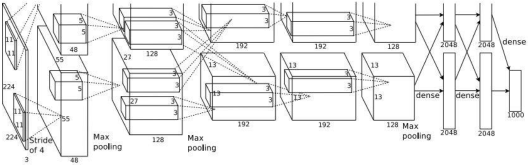
*[Krizhevsky et al. 2012]*




Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

# Case Study: AlexNet

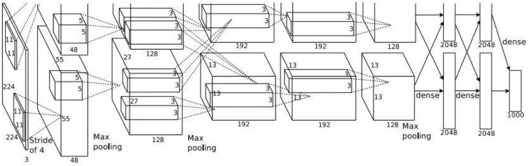
*[Krizhevsky et al. 2012]*




Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

# Case Study: AlexNet

*[Krizhevsky et al. 2012]*

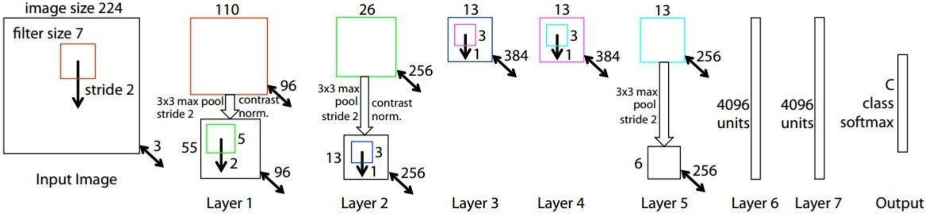



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

**Details/Retrospectives:**
- first use of ReLU
- used Norm layers (not common anymore)
- data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
-Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4

# Case Study: ZFNet

*[Zeiler and Fergus, 2013]*




Same as AlexNet architecture but:

* CONV1: change from (11x11 stride 4) to (7x7 stride 2)

The reasoning behind this modification is that a smaller filter size in the first conv layer helps retain a lot of original pixel information in the input volume.

* CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512
* Developed a visualization technique named Deconvolutional Network, which helps to examine different feature activations and their relation to the input space.
* Trained on a GTX 580 GPU for **twelve days**.

# Case Study: VGGNet

*[Simonyan and Zisserman, 2014]*

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

| 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<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Table 2: **Number of parameters** (in millions).

| Network | A,A-LRN | B | C | D | E |
|---|---|---|---|---|---|
| Number of parameters | 133 | 133 | 134 | 138 | [illegible] |

Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.

# Case Study: VGGNet

*[Simonyan and Zisserman, 2014]*

(not counting biases)

NPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

| ConvNet Configuration | | | |
|---|---|---|---|
| B | C | D | |
| 13 weight layers | 16 weight layers | 16 weight layers | 19 |
| put (224 × 224 RGB image | | | |
| conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | co<br>co |
| maxpool | | | |
| conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | co<br>co |
| maxpool | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| FC-4096 | | | |
| FC-4096 | | | |
| FC-1000 | | | |
| soft-max | | | |

# Case Study: VGGNet

*[Simonyan and Zisserman, 2014]*

(not counting biases)

NPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

| ConvNet Configuration | | | |
|---|---|---|---|
| B | C | D | |
| 13 weight layers | 16 weight layers | 16 weight layers | 19 |
| put (224 × 224 RGB image | | | |
| conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | co<br>co |
| maxpool | | | |
| conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | co<br>co |
| maxpool | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | co<br>co<br>co<br>**co** |
| maxpool | | | |
| FC-4096 | | | |
| FC-4096 | | | |
| FC-1000 | | | |
| soft-max | | | |

# Case Study: VGGNet

*[Simonyan and Zisserman, 2014]*

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

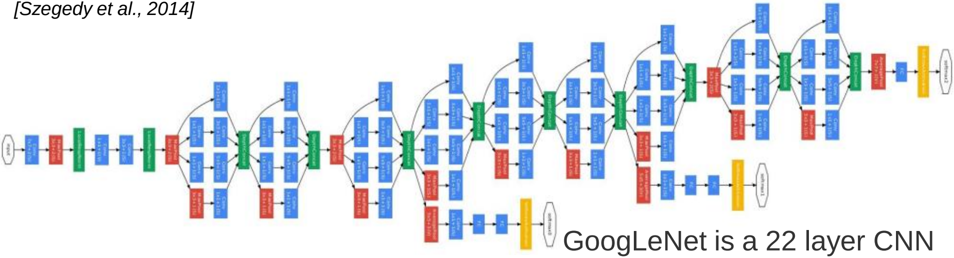
Most memory is in early CONV

Most params are in late FC

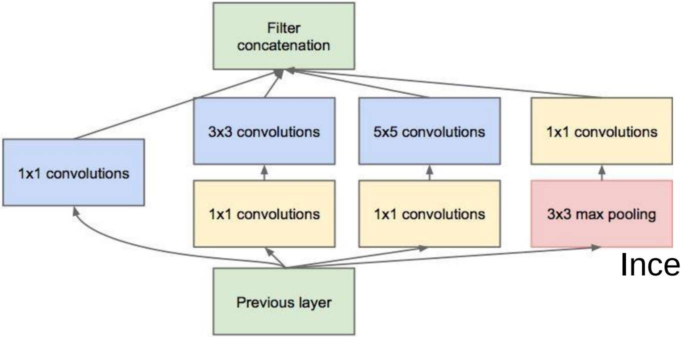
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

# Case Study: GoogLeNet

*[Szegedy et al., 2014]*



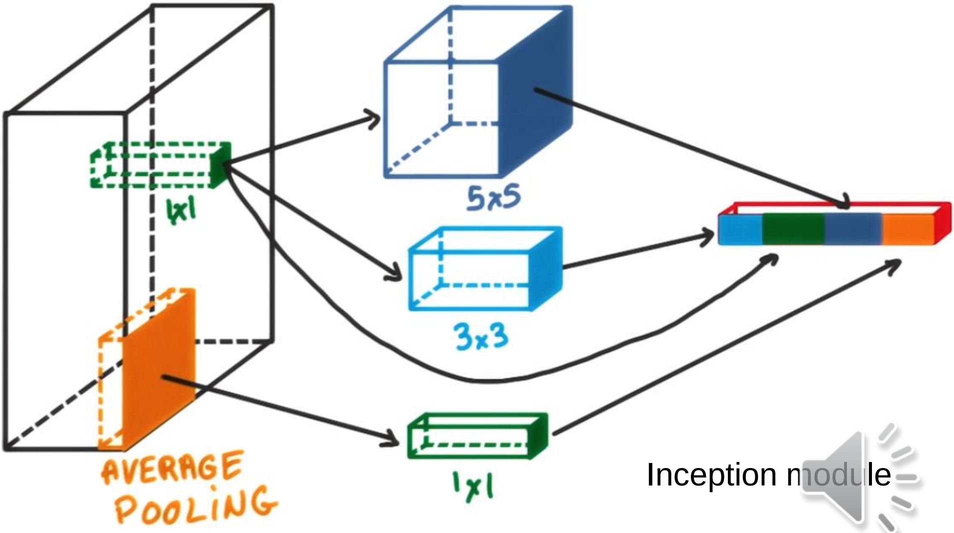
GoogLeNet is a 22 layer CNN




Inception module

# Case Study: GoogLeNet

*[Szegedy et al., 2014]*




Inception module

# Case Study: GoogLeNet

- Used 9 Inception modules in the whole architecture, with over 100 layers in total!
- No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet. (5M params), but 2x more compute.
- There are updated versions to the Inception model (**Inception-V2, Inception-V3, Inception-V4**) .
- Trained on “a few high-end GPUs within a week”.

# Case Study: ResNet

*[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)

Microsoft Research

## MSRA @ ILSVRC & COCO 2015 Competitions

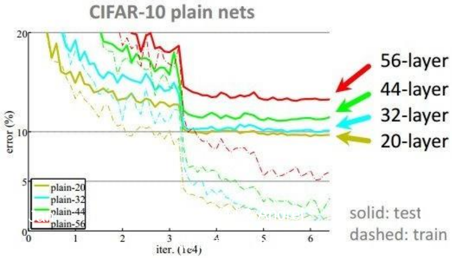
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

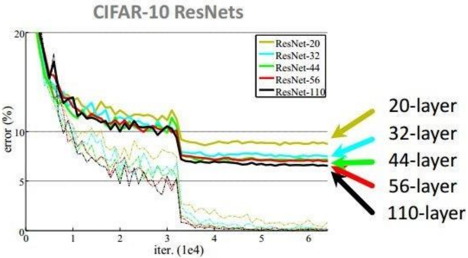
*improvements are relative numbers

ICCV15

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

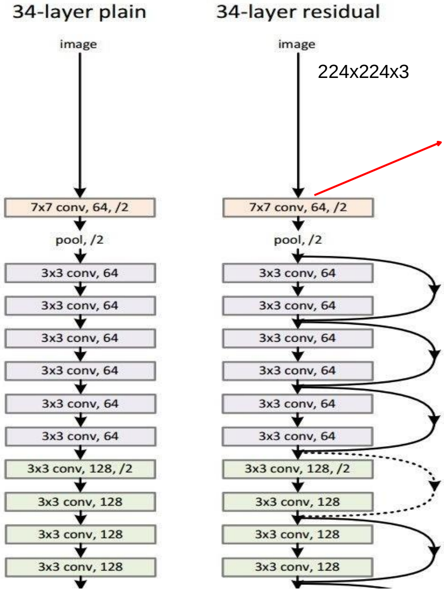
# CIFAR-10 experiments



# Case Study: ResNet

*[He et al., 2015]*



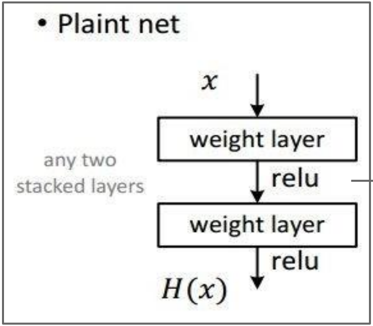

spatial dimension only 56x56!

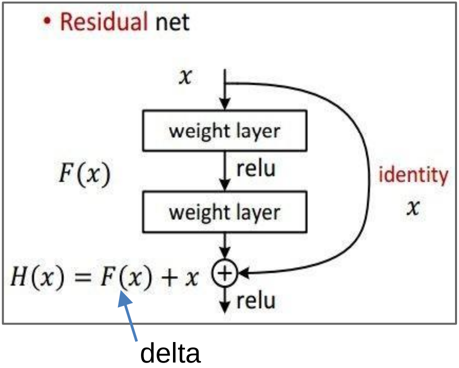
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

# Case Study: ResNet

*[He et al., 2015]*



# Other CNN models

1. ResNeXt.
3. MobileNet V1/V2.
4. DenseNet.
5. Xception model.
6. NASNet
7. Inception-ResNet

# Pre-trained Models & Transfer Learning

# Pre-trained Model

- In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.

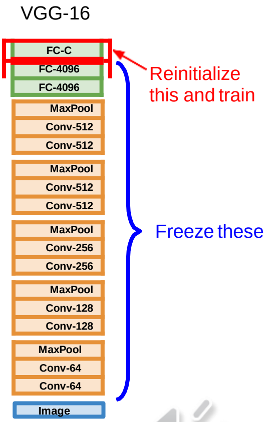
- Instead, a pre-trained model is used. A pre-trained model is a saved network that was previously trained on a large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), as AlexNet, VGG, etc.… The network has learned rich feature representations for a wide range of images.

- You either use the pre-trained model as is or use transfer learning to customize this model to a given task.

# Transfer Learning

Transfer Learning scenarios look as follows:

**1- ConvNet as fixed feature extractor**
Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. We call these features **CNN codes**.
Once CNN codes are extracted for all images, train a new classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.



# Transfer Learning (Cont.)

Transfer Learning scenarios look as follows:

**2- Fine-tuning the ConvNet**. The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation, by unfreeze some layers.

It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes more specific to the details of the original dataset.

# When and how to fine-tune?

This is a function of several factors, but the two most important ones are:

1- The **size of the new dataset** (small or big)

2- **Dataset similarity to the original dataset**, e.g. ImageNet-like in terms of the content of images and the classes, or very different, such as microscope images.

# Rules of Thumb

Some common rules of thumb for navigating the 4 major scenarios:

***1- New dataset is small and similar to original dataset*.**
Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.

***2- New dataset is large and similar to the original dataset*.**
Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune more layers than previous case.

# Rules of Thumb (Cont.)

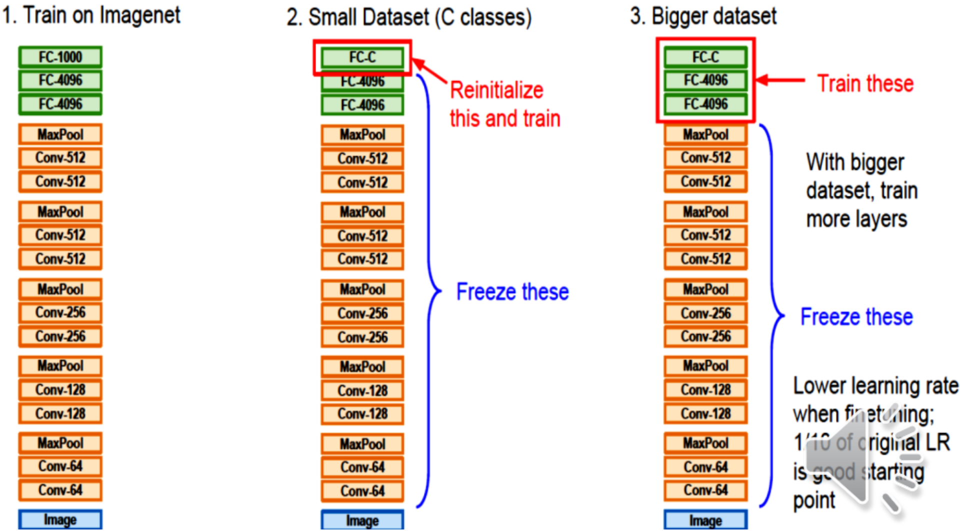
*3-* ***New dataset is small but very different from the original dataset*.**
Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.

*4-* ***New dataset is large and very different from the original dataset*.**
Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

# In Summary

## Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



# Model Ensembles

We obtained a fairly good accuracy with **Transfer Learning**, but we still weren't satisfied. So we decided to use all the models we trained at the same time.

1- Train multiple independent models
2- **Hard Voting** performs a simple majority vote taking predicted classes into consideration, or **Soft voting** takes an average of the probabilities predicted by each model for each class, selecting the most likely one in the end.

Enjoy 2% extra performance

# This lecture references

- CS231 Stanford:https://www.youtube.com/watch?v=LxfUGhug-iQ

- https://cs231n.github.io/transfer-learning/
- https://www.tensorflow.org/tutorials/images/transfer_learning
- https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html
- https://towardsdatascience.com/deep-blue-sea-using-deep-learning-to-detect-hundreds-of-different-plankton-species-dff895d3b226