Convolutional Neural Networks

Week 2

Outline

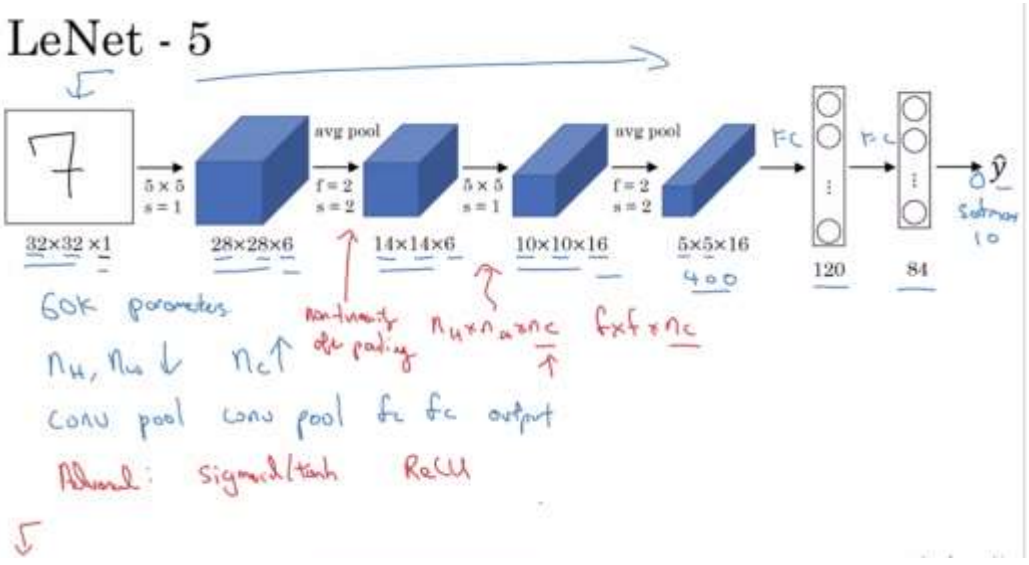
Classic networks:

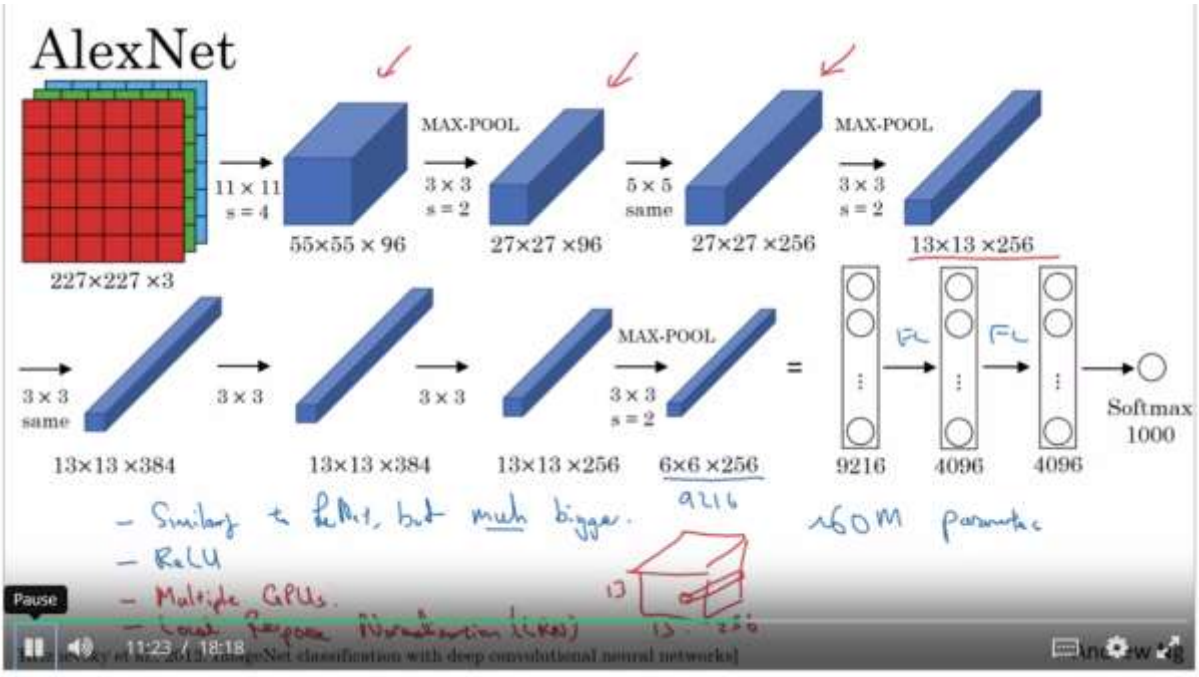
- LeNet-5 ⇐
- AlexNet ⇐
- VGG ⇐

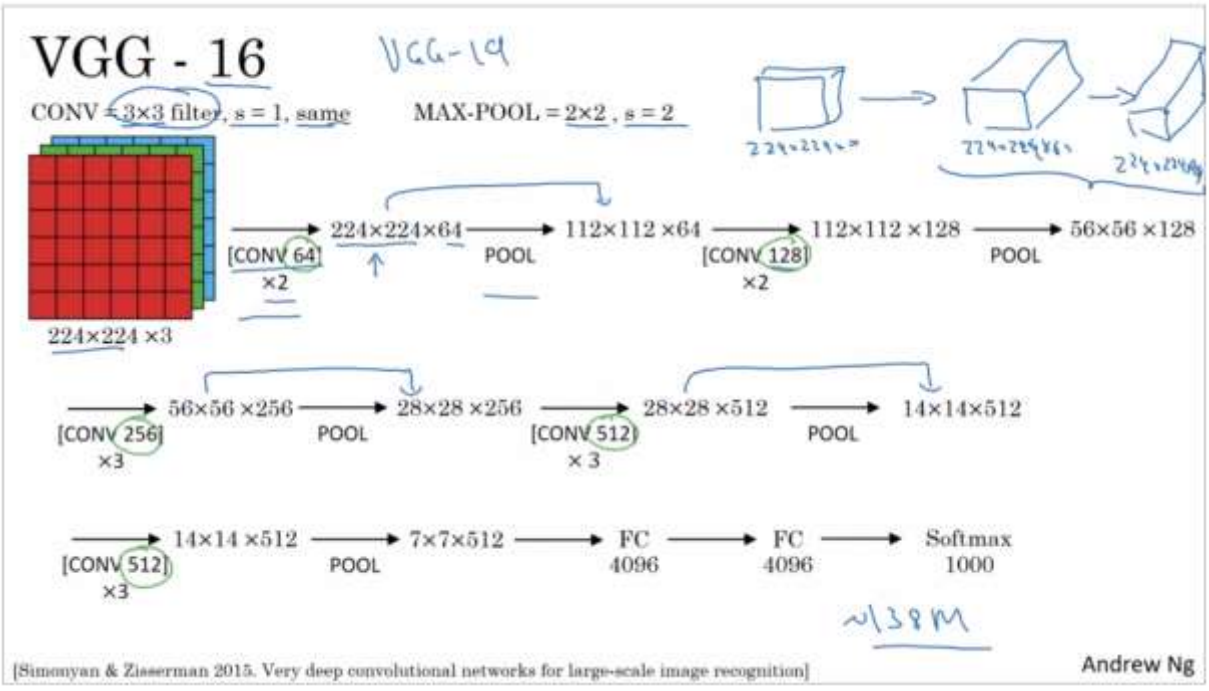
ResNet (152)

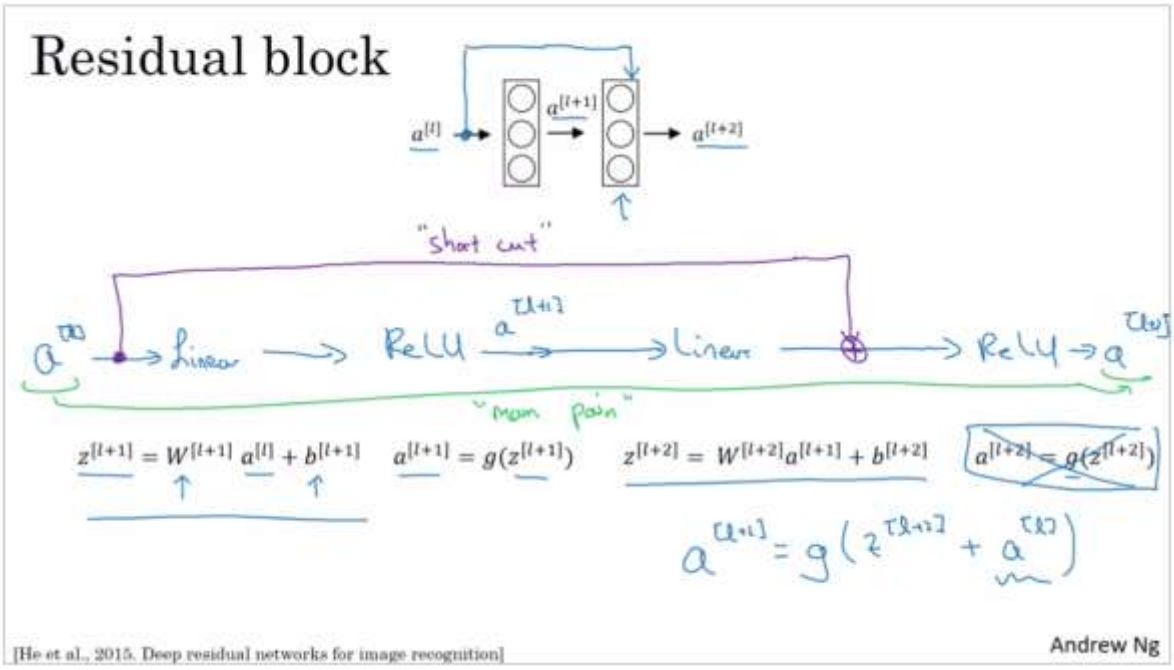
Inception

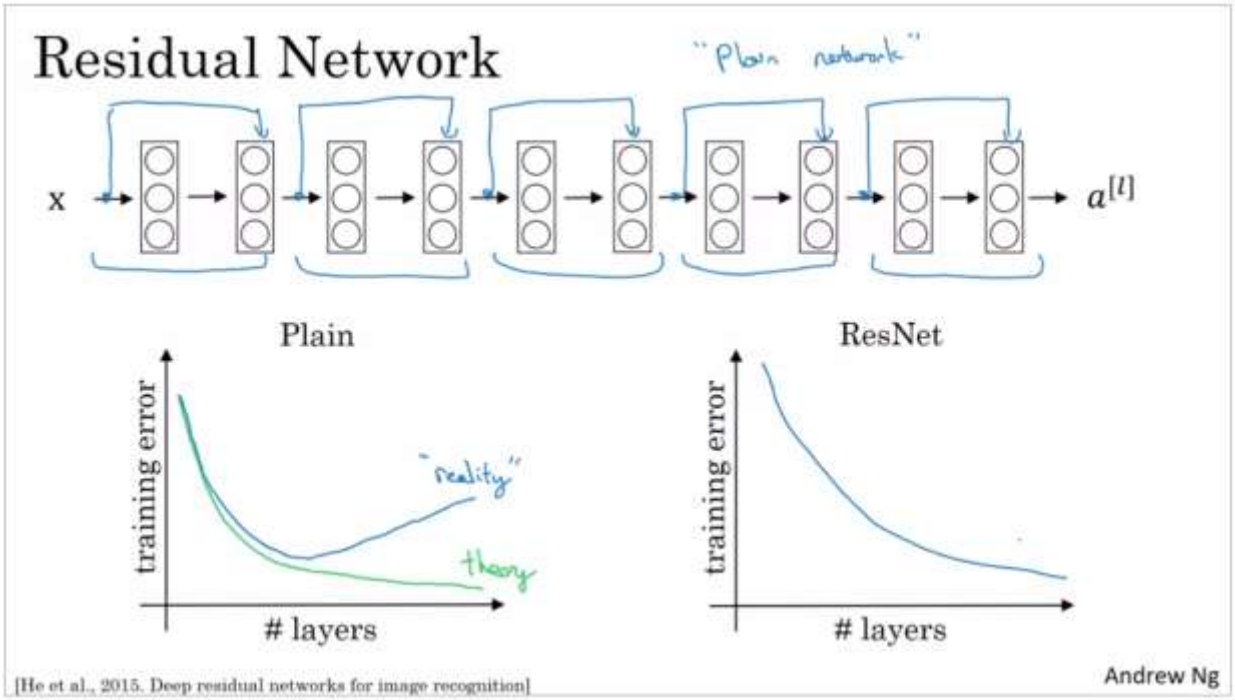
LeNet - 5



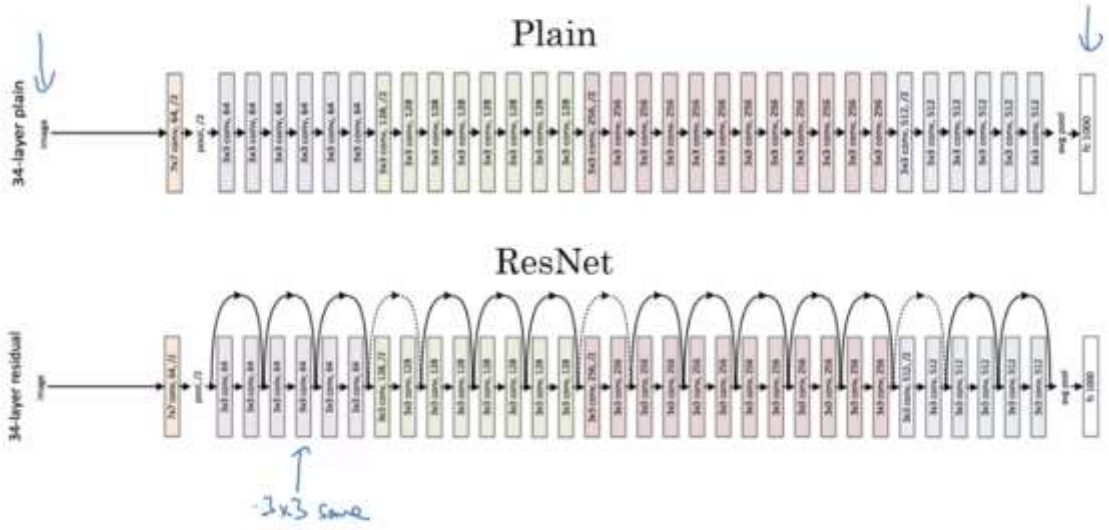

AlexNet
227×227 ×3
11 × 11
s = 4
55×55 × 96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6×6 ×256
=
9216
FC
4096
FC
4096
Softmax
1000
- Similar to LeNet, but much bigger.
- ReLU
- Multiple GPUs.
- Local response Normalization (LRN)
60M parameters
Pause
11:23 / 18:18
[Krizhevsky et al. 2012. ImageNet classification with deep convolutional neural networks]


VGG - 16
VGG-16
CONV = 3×3 filter, s = 1, same
MAX-POOL = 2×2 , s = 2
224×224 ×3
[CONV 64]
×2
224×224×64
POOL
112×112 ×64
[CONV 128]
×2
112×112 ×128
POOL
56×56 ×128
[CONV 256]
×3
56×56 ×256
POOL
28×28 ×256
[CONV 512]
× 3
28×28 ×512
POOL
14×14×512
[CONV 512]
×3
14×14 ×512
POOL
7×7×512
FC
4096
FC
4096
Softmax
1000
~138M
[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]
Andrew Ng

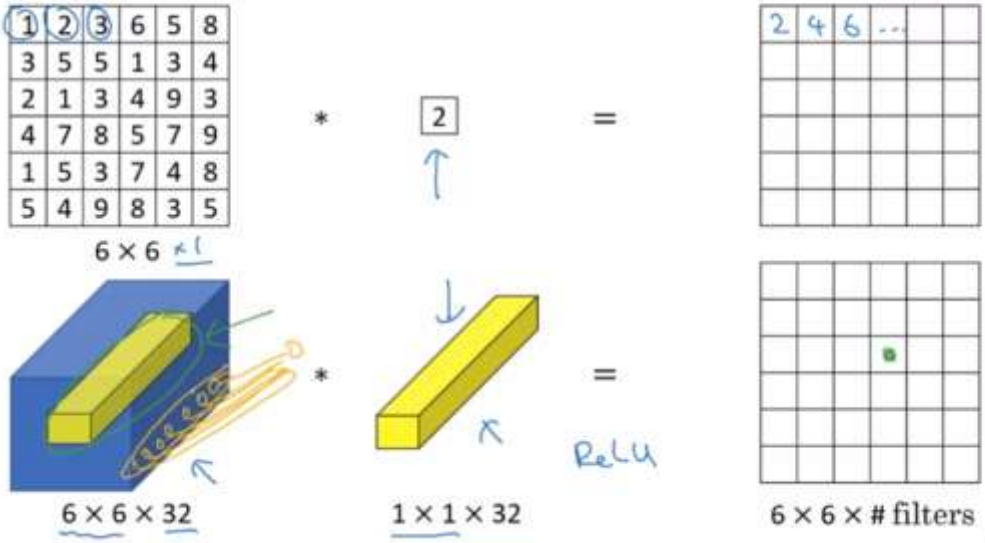

Residual block
$a^{[l]}$
$a^{[l+1]}$
$a^{[l+2]}$
"Short cut"
Linear
ReLU
Linear
ReLU
"Main Path"
$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$
$a^{[l+1]} = g(z^{[l+1]})$
$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$
$a^{[l+2]} = g(z^{[l+2]})$
$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$
[He et al., 2015. Deep residual networks for image recognition]
Andrew Ng


Residual Network
"Plain network"
x
$a^{[l]}$
Plain
ResNet
training error
"reality"
theory
layers
training error
layers
[He et al., 2015. Deep residual networks for image recognition]
Andrew Ng

ResNet

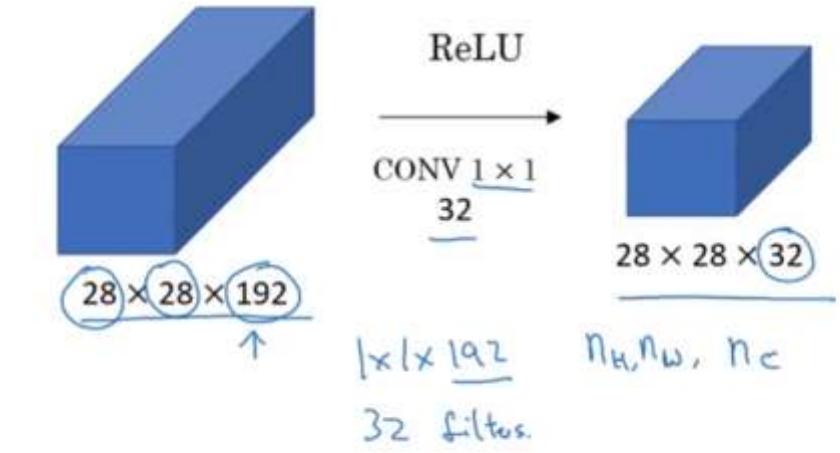


Why does a 1 × 1 convolution do?

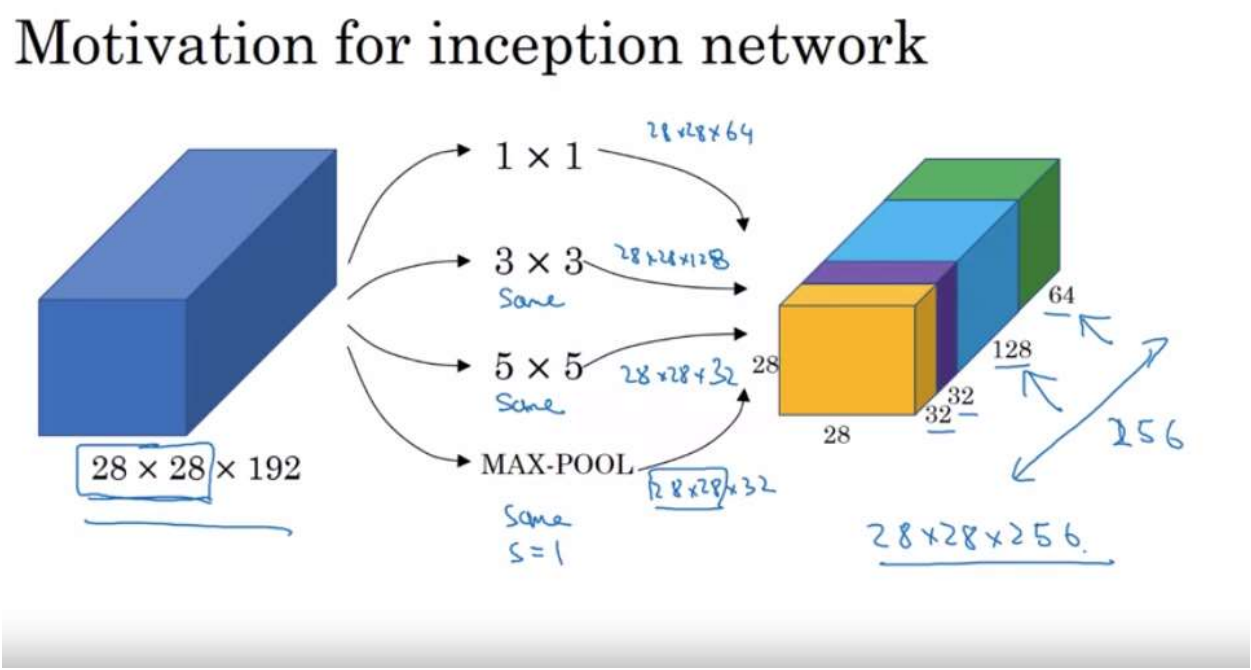


[Lin et al., 2013. Network in network]

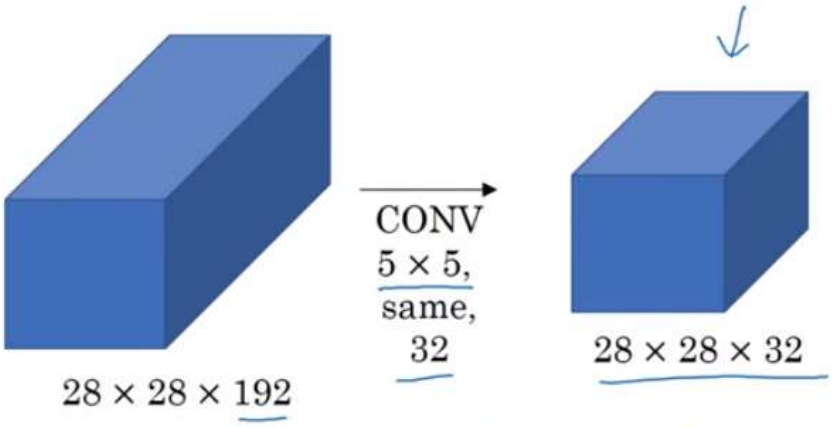
Andrew Ng



Using the Network in Network technique allows us to control the number of channels. (It is used in Inception Networks)



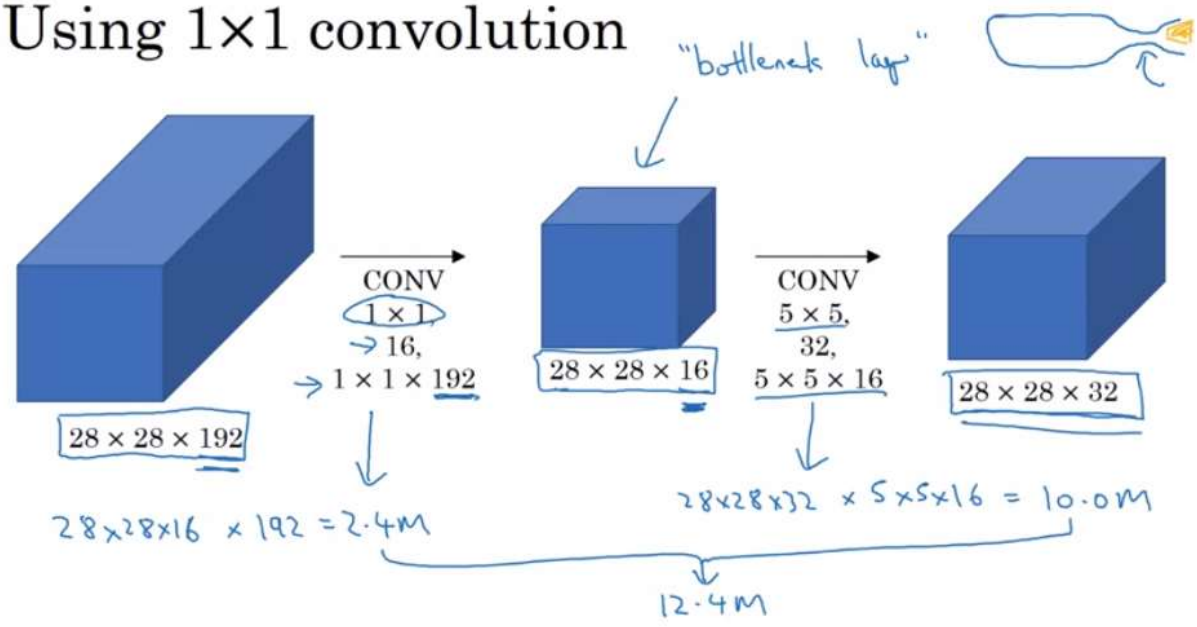
The problem of computational cost



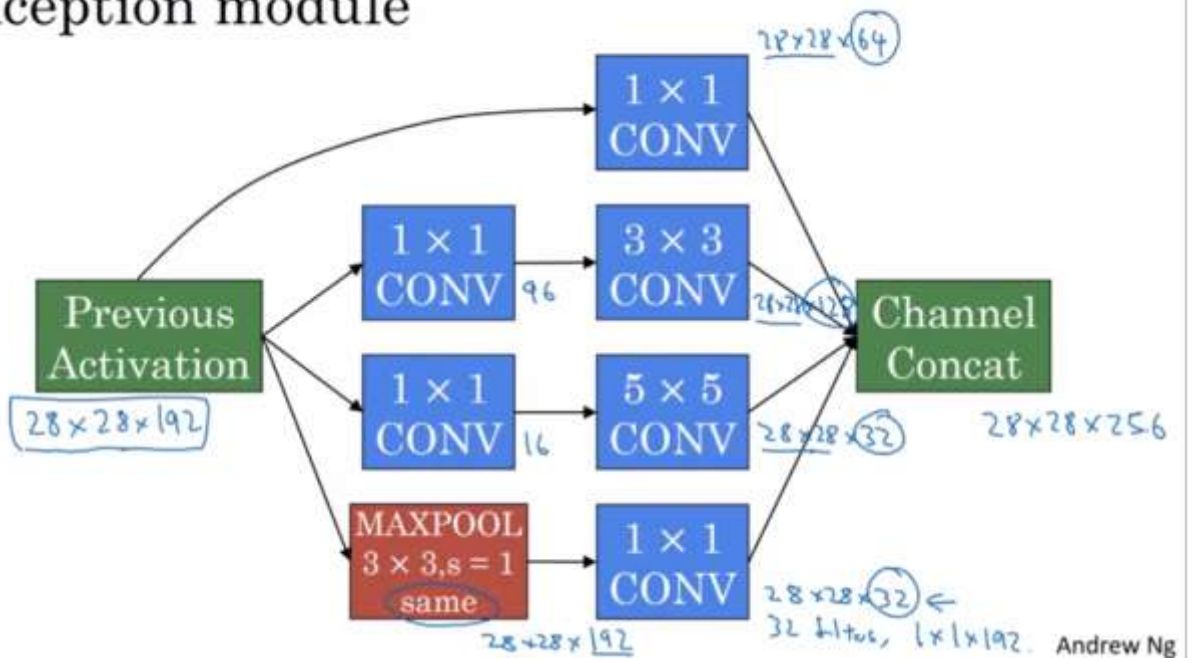
32 filters. filter one 5×5×192.

28×28×32 × 5×5×192 = 120M.

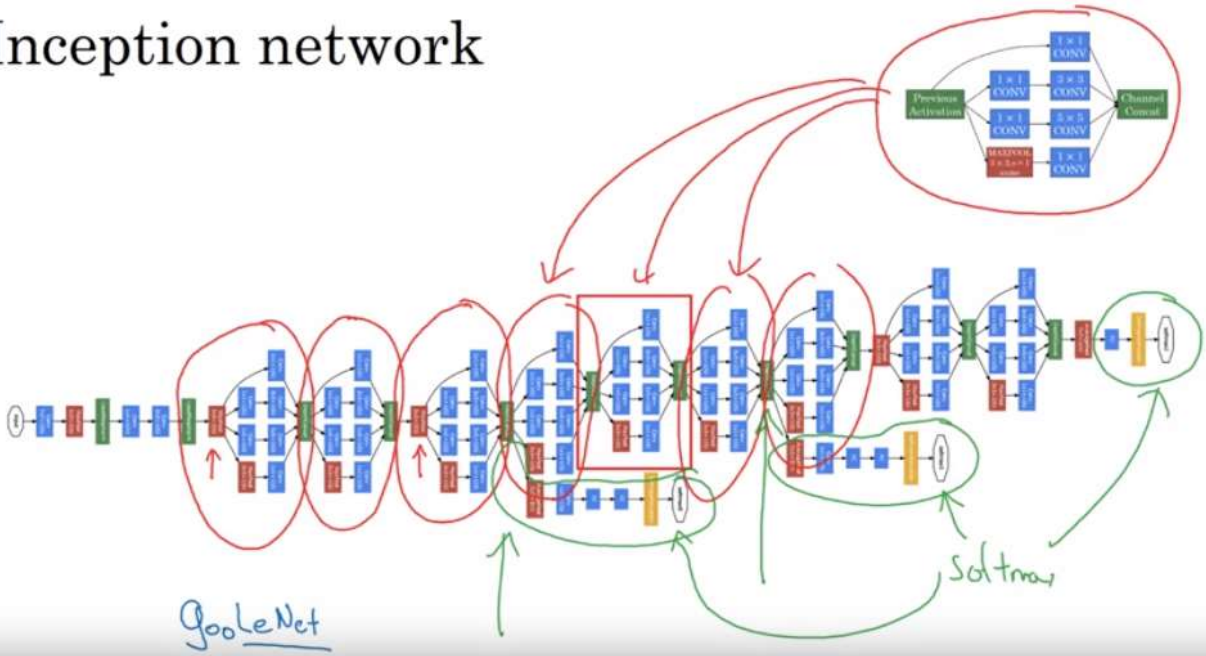
Using 1×1 convolution



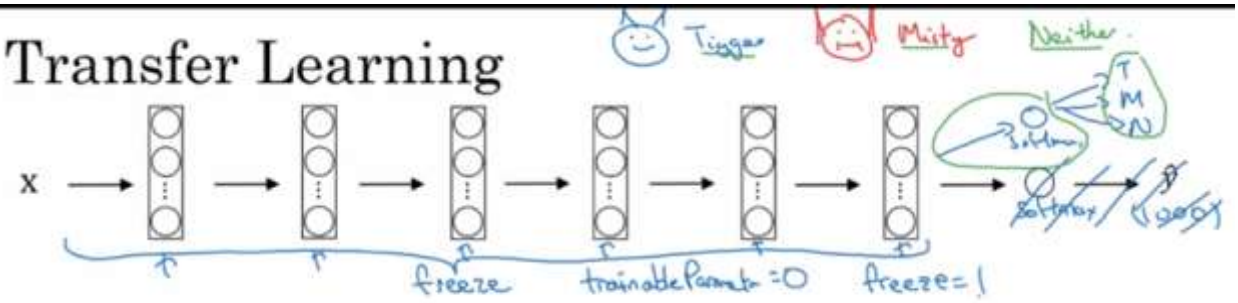
Inception module



Inception network



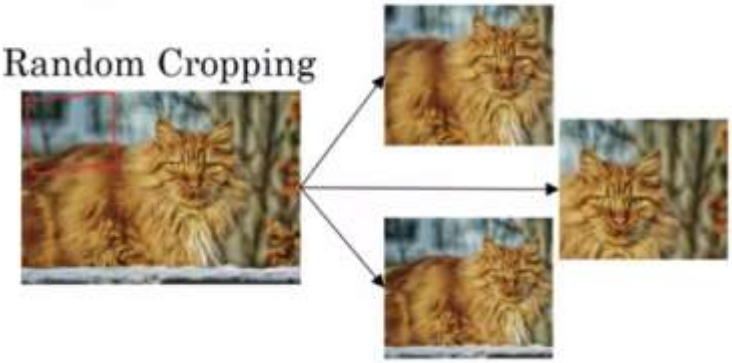
Transfer Learning



Common augmentation method

Mirroring

Random Cropping



Rotation

Shearing

Local warping

...

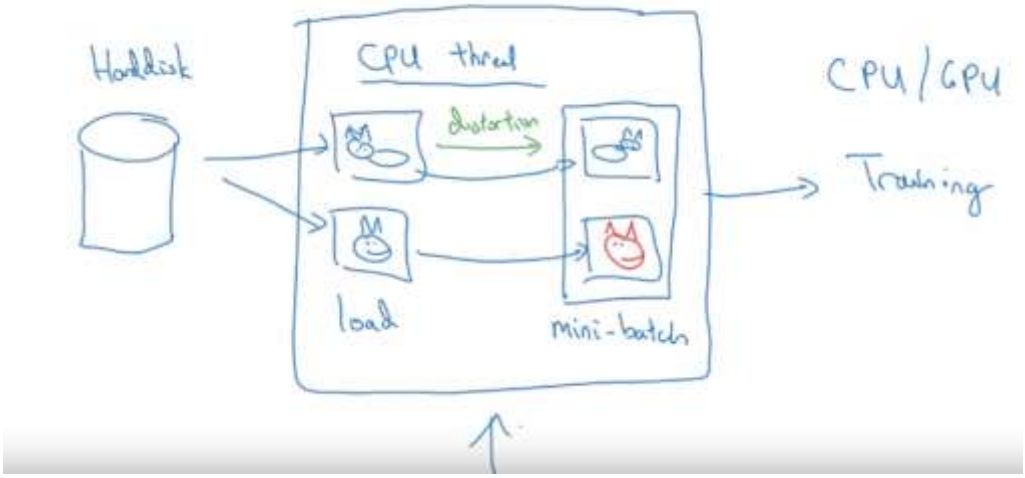
Andrew Ng

Color shifting

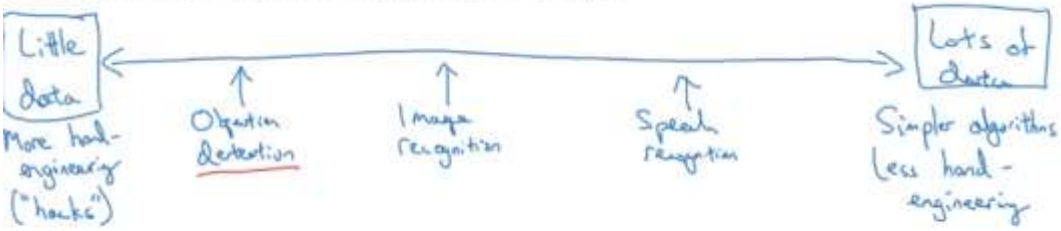
Advanced:
PCA
ml-class.org
AlexNet paper
"PCA color augmentation."
R B G

Andrew Ng

Implementing distortions during training



Data vs. hand-engineering



Two sources of knowledge

- Labeled data (x, y)
- Hand engineered features/network architecture/other components

Andrew Ng

Tips for doing well on benchmarks/winning competitions

Ensembling

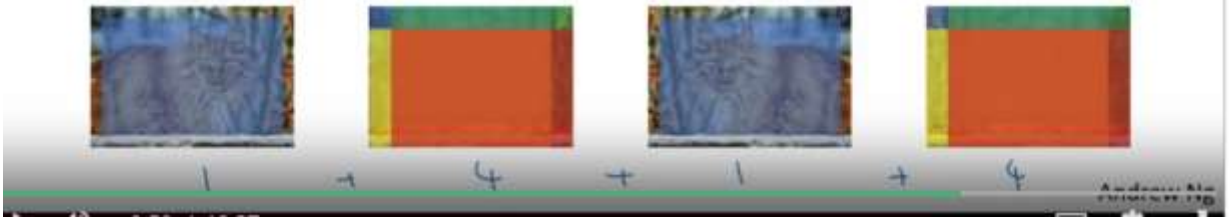
3-15 networks → $\hat{y}$

- Train several networks independently and average their outputs

Multi-crop at test time

- Run classifier on multiple versions of test images and average results

10-crop



Why are we using Keras?

- Keras was developed to enable deep learning engineers to build and experiment with different models very quickly.
- Just as TensorFlow is a higher-level framework than Python, Keras is an even higher-level framework and provides additional abstractions.
- Being able to go from idea to result with the least possible delay is key to finding good models.
- However, Keras is more restrictive than the lower-level frameworks, so there are some very complex models that you would still implement in TensorFlow rather than in Keras.
- That being said, Keras will work fine for many common models.

```
def model(input_shape):
    """
    input_shape: The height, width and channels as a tuple.
        Note that this does not include the 'batch' as a dimension.
        If you have a batch like 'X_train',
        then you can provide the input_shape using
        X_train.shape[1:]
    """

    # Define the input placeholder as a tensor with shape input_shape. Think of this as your input image!
    X_input = Input(input_shape)

    # Zero-Padding: pads the border of X_input with zeroes
    X = ZeroPadding2D((3, 3))(X_input)

    # CONV -> BN -> RELU Block applied to X
    X = Conv2D(32, (7, 7), strides = (1, 1), name = 'conv0')(X)
    X = BatchNormalization(axis = 3, name = 'bn0')(X)
    X = Activation('relu')(X)

    # MAXPOOL
    X = MaxPooling2D((2, 2), name='max_pool')(X)

    # FLATTEN X (means convert it to a vector) + FULLYCONNECTED
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', name='fc')(X)

    # Create model. This creates your Keras model instance, you'll use this instance to train/test the model.
    model = Model(inputs = X_input, outputs = X, name='HappyModel')
```

Objects as functions (a feature of Python):

```
ZP = ZeroPadding2D((3, 3)) # ZP is an object that can be called as a function
X = ZP(X_input)
```

Tips for improving your model

If you have not yet achieved a very good accuracy (>= 80%), here are some things tips:

- Use blocks of CONV->BATCHNORM->RELU such as:

```
X = Conv2D(32, (3, 3), strides = (1, 1), name = 'conv0')(X)
X = BatchNormalization(axis = 3, name = 'bn0')(X)
X = Activation('relu')(X)
```

  until your height and width dimensions are quite low and your number of channels quite large (≈32 for example).
  You can then flatten the volume and use a fully-connected layer.
- Use MAXPOOL after such blocks. It will help you lower the dimension in height and width.
- Change your optimizer. We find 'adam' works well.
- If you get memory issues, lower your batch_size (e.g. 12)
- Run more epochs until you see the train accuracy no longer improves.

Note: If you perform hyperparameter tuning on your model, the test set actually becomes a dev set, and your model might end up overfitting to the test (dev) set. Normally, you'll want separate dev and test sets. The dev set is used for parameter tuning, and the test set is used once to estimate the model's performance in production.

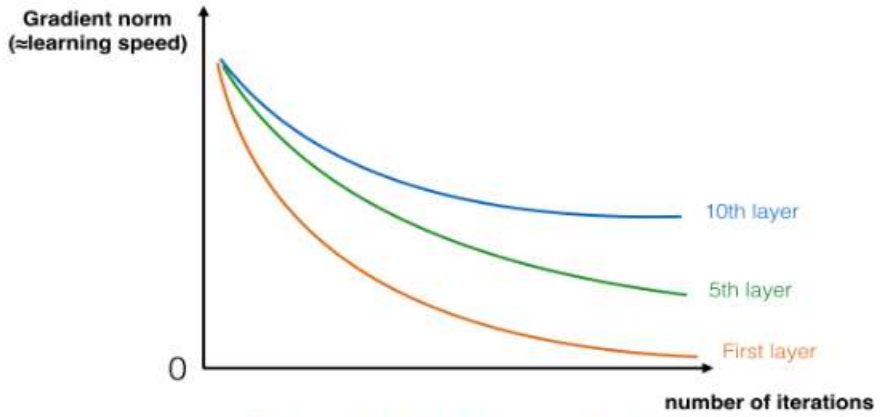


Figure 1 : **Vanishing gradient**

The speed of learning decreases very rapidly for the shallower layers as the network trains

Introduction of Residual Networks is one way to tackle this problem.

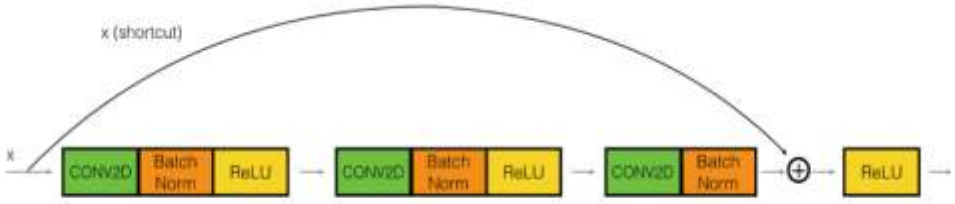


Figure 4 : **Identity block.** Skip connection "skips over" 3 layers.

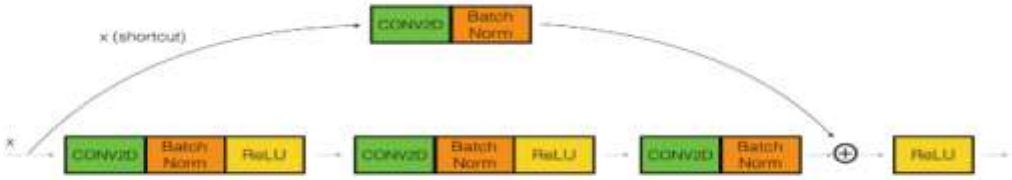


Figure 4 : **Convolutional block**

You now have the necessary blocks to build a very deep ResNet. The following figure describes in detail the architecture of this neural network. "ID BLOCK" in the diagram stands for "identity block," and "ID BLOCK x3" means you should stack 3 identity blocks together.

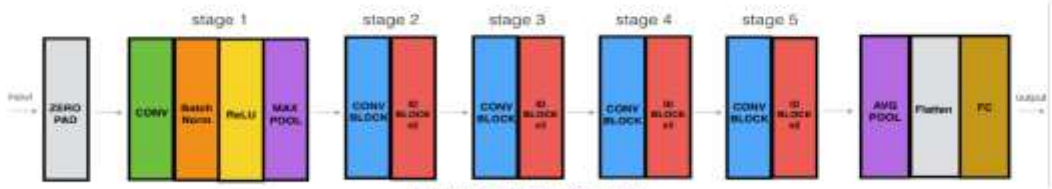


Figure 5 : **ResNet-50 model**

- Very deep "plain" networks don't work in practice because they are hard to train due to vanishing gradients.
- The skip-connections help to address the Vanishing Gradient problem. They also make it easy for a ResNet block to learn an identity function.
- There are two main types of blocks: The identity block and the convolutional block.
- Very deep Residual Networks are built by stacking these blocks together.