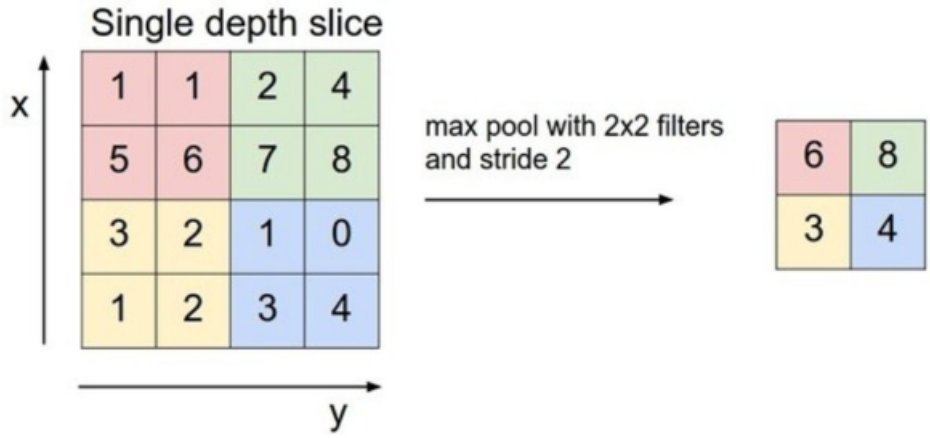
Review: CNN Architecture

CNN을 구성하는 layers

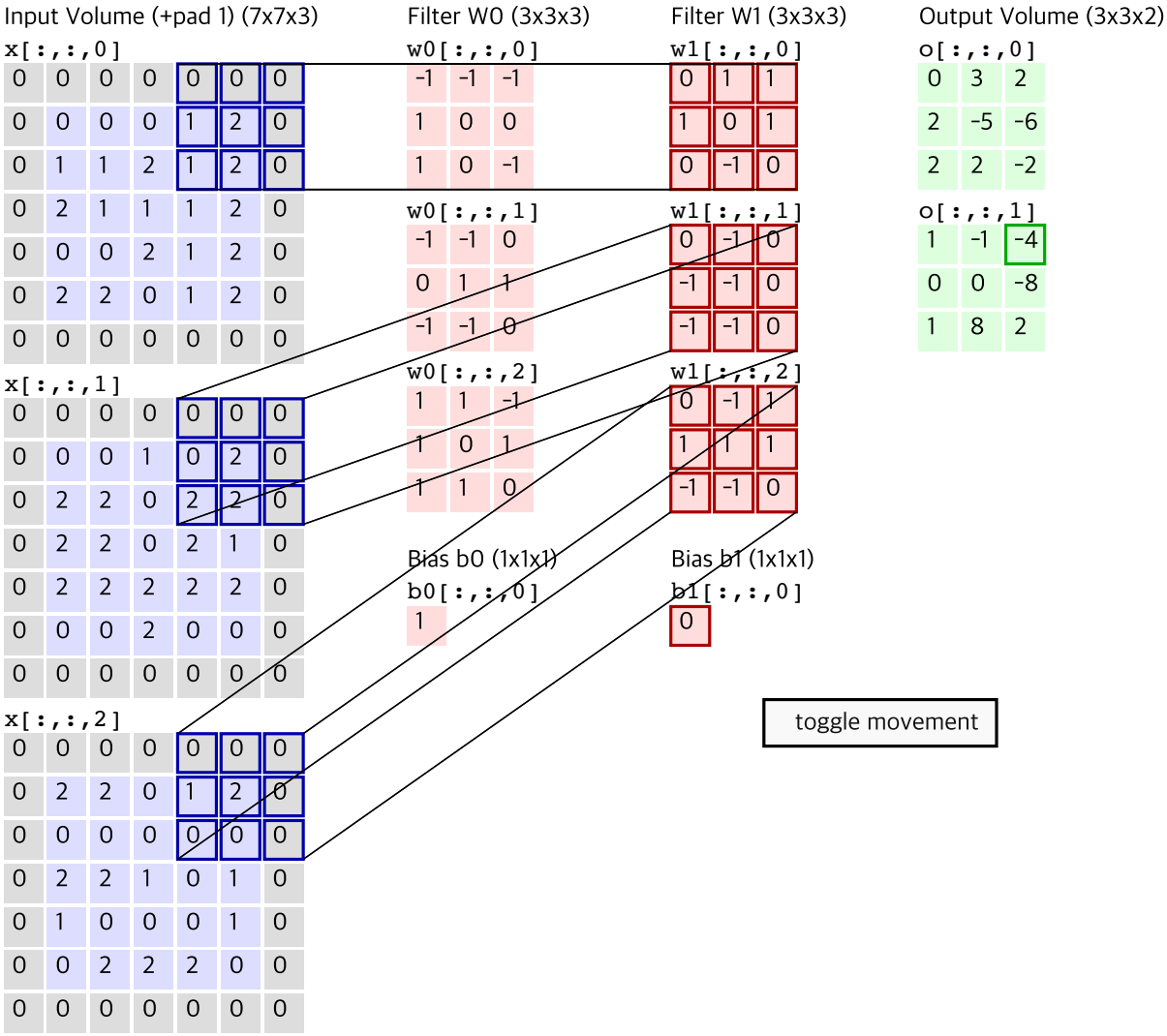
- Input layer
  - 입력 이미지가 가로32, 세로32, 그리고 RGB 채널을 가지는 경우 입력의 크기는 [32x32x3]
- Convolution layer
  - 각 convolution operation은 filter를 통해 input의 일부분과 연결
  - 만일 12개의 filter를 사용한다면 [32(?)x32(?)x12] 크기의 출력 생성
  - 일반적으로 ReLU activation과 함께 사용
- Pooling layer
  - downsampling을 통해 계산량을 줄이고 특성을 강화
  - [32x32x12]를 입력받았다면 [16x16x12]를 출력
  - 학습할 parameter가 없음



- Fully-connected layer
  - Input의 각 뉴런들과 모두 연결되어 있는 레이어
  - 만일 노드가 10개라면 [10x1] 크기를 출력
  - 일반적으로 ReLU activation과 함께 사용

Convolution operation

In [1]:
```
from IPython.display import IFrame
IFrame(src='https://cs231n.github.io/assets/conv-demo/index.html', width=800, height=
```



- Input: $W_1 \times H_1 \times D_1$ 크기를 입력으로 받음
- Hyperparameters
  - Filter의 수: K
  - Filter의 크기: F
  - Stride(몇 칸씩 건너뛰며 convolution operation을 하는가?): S
  - Zero padding의 수: P
- Output: $W_2 \times H_2 \times D_2$ 크기가 output으로 나옴
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$

Covolution architecture의 패턴

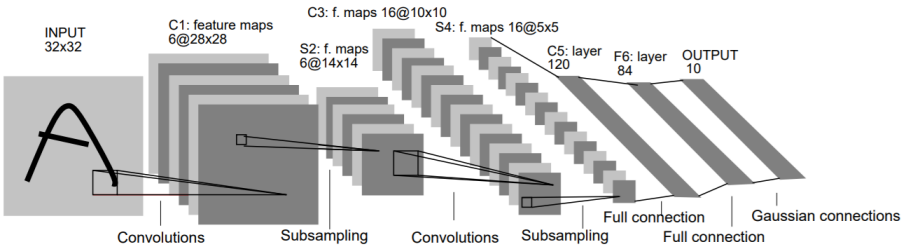
일반적으로 convolution(CONV), pooling(POOL), fully connected(FC) layer로 구성되어 있음

- **INPUT -> FC**
  - 선형 분류기
- **INPUT -> CONV -> FC**
- **INPUT -> [CONV -> POOL]*2 -> FC -> FC**
  - CONV 하나와 POOL 하나를 연결하는 세트를 두 번 반복
  - FC 하나 연결 후 output layer로 연결
- **INPUT -> [CONV -> CONV -> POOL]*3 -> FC *2 -> FC**
  - CONV 두 개와 POOL 하나를 연결하는 세트를 세 번 반복
  - POOL 연산으로 정보가 손실되기 전에 여러 층으로 쌓인 CONV 레이어를 통해 복잡한 feature들을 추출

Lecture 5. Classic Convolutional Neural Networks

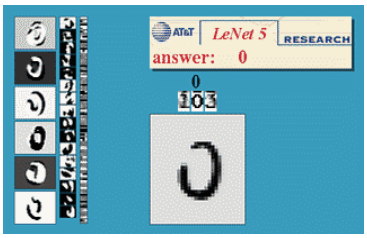
LeNet (1998)

- Yann LeCunn et al. (http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf)(NYU)에 의해 개발
- CNN 개념의 시초
- Handwritten digit recognition (MNIST)에 널리 사용
- Convolution을 거치면서 작은 크기의 feature에서 보다 일반화된 feature를 얻어가는 과정을 제시
- Architecture



Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

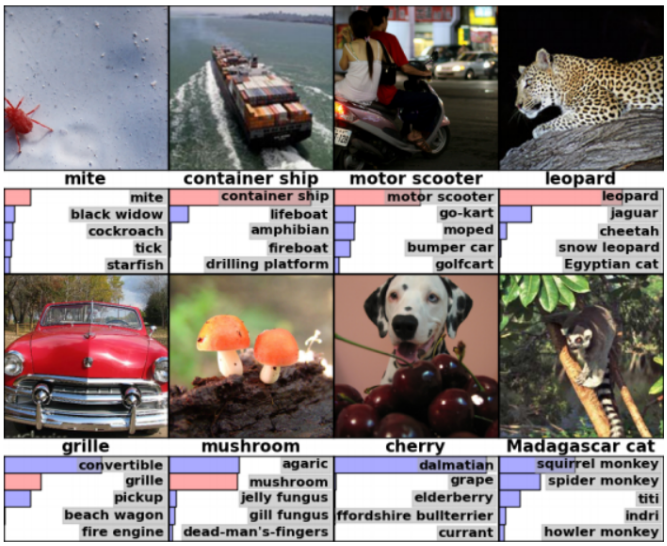
  - 약 60,000개의 parameter
  - 28×28 크기인 MNIST input을 padding=2를 사용하여 32×32로 만들어 input 이미지 구성
  - Average pooling 사용. 현재는 max pooling을 더 많이 사용
  - Output layer에서 radial basis function을 사용. 현재는 softmax를 더 많이 사용



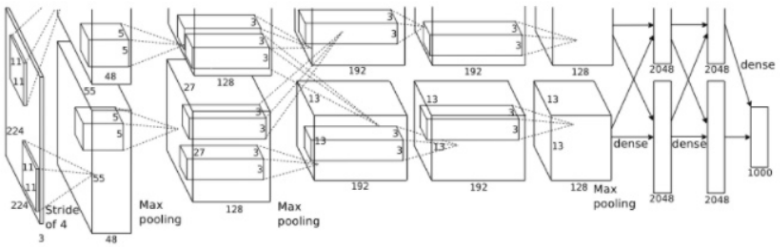
http://yann.lecun.com/exdb/lenet/index.html

AlexNet (2012)

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton(U of Toronto)에 의해 개발 http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)
- 2012 ImageNet ILSVRC challenge에서 우승
- 1000개의 범주를 가지는 1500만 개의 image classification



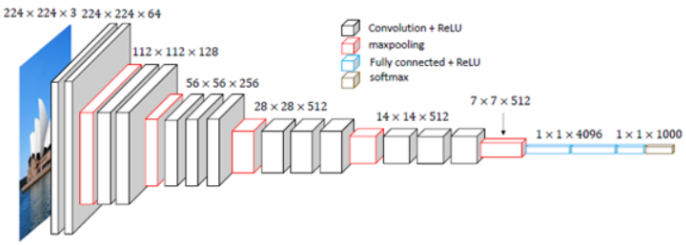
- Architecture



  - LeNet과 비슷하지만 훨씬 깊고 큰 network
  - 5개의 convolution layer, 3개의 fully-connected layer 사용
  - GPU 두 개를 기반으로한 병렬 구조
  - ReLU activation을 사용
  - Dropout의 소개
  - 약 6천만 개의 parameter

VGGNet (2014)

- ImageNet ILSVRC Challenge 2014에서 Simonyan and Zisserman(Oxford Univ.)에 의해 제안(2등) http://www.robots.ox.ac.uk/~vgg/research/very_deep/ (http://www.robots.ox.ac.uk/~vgg/research/very_deep/)
- 단순한 구조로 지금까지 많이 사용, 망의 구조보다는 망의 깊이가 중요하다는 것을 보여줌 (16개 layer 이상이면 큰 개선 없음; VGG16이 가장 좋은 성능)
- 항상 3×3 filter, Stride=1, same padding, 2×2 pooling 사용
- 2014년 1등인 GoogLeNet 보다 이미지 분류 성능은 낮지만 transfer learning 과제에서 더 좋은 성능을 보이는 경우가 많음
- Architecture



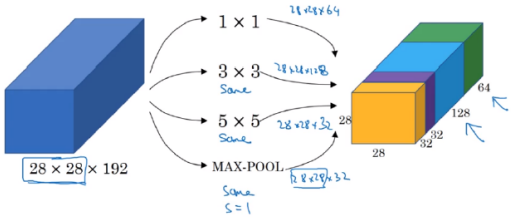
  - Filter의 수가 64, 128, 256, ... 두 배씩 커짐
  - 약 1억3천8백만 개의 parameter: 매우 많은 메모리와 연산량이 필요

GoogLeNet(2014)

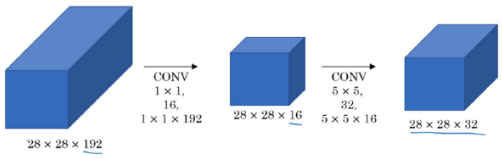
- ImageNet ILSVRC Challenge 2014의 우승 모델. Google의 Szegedy et al.이 제안
- Parameter의 수를 획기적으로 줄여주는 inception module을 소개

Inception module

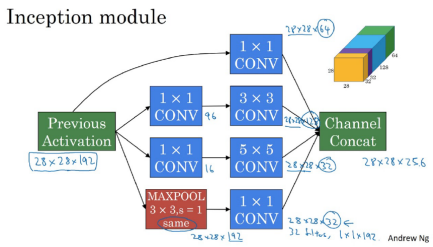
- 어느 사이즈의 필터를 사용할 것인가?
  - $1 \times 1, 3 \times 3, 5 \times 5$filter를 모두 사용하여 쌓고 weight를 학습시킴



- Bottleneck layer
  - 계산량이 너무 많은 단점을 해결하기 위해 1×1 filter를 사용하여 사이즈를 줄여준 뒤 5×5 filter 적용

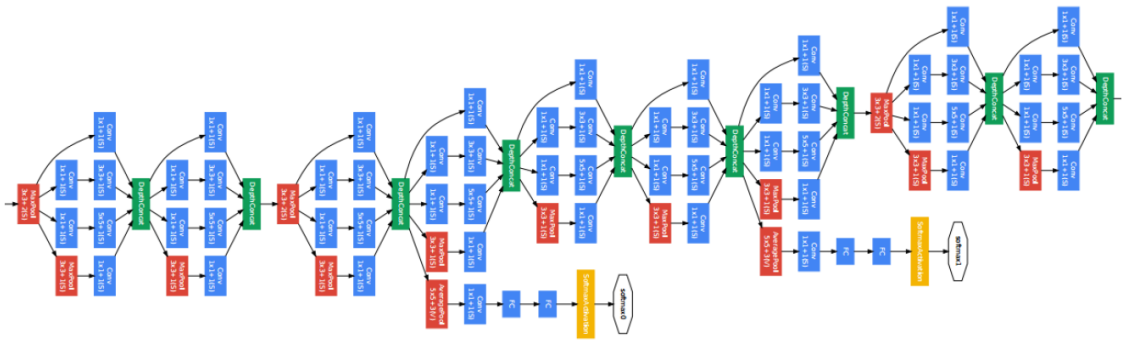


- Inception module
  - $(1 \times 1, 3 \times 3), (1 \times 1, 5 \times 5)$를 사용하여 stack



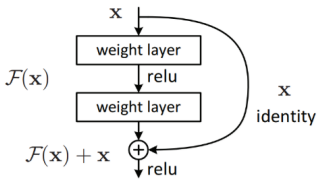
GoogLeNet architecture

- Inception module의 반복적용
- 중간에 maxpooling으로 사이즈 축소
- Hidden layer에서 softmax를 통해 output을 예측하는 과정을 추가하여 layer 전체를 통과하지 않더라도 나쁘지 않은 예측을 가능하도록 설계(overfitting 방지의 역할)

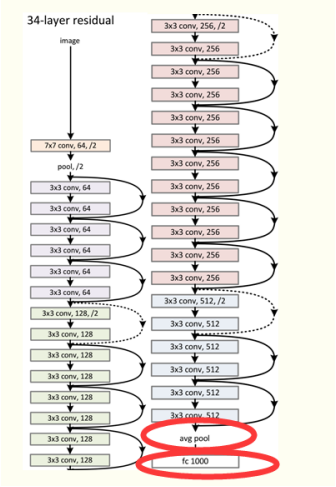


ResNet (2015)

- ImageNet ILSVRC 2015 우승 (error rate 3.6%, 인간의 error rate 5~10%)
- Kaiming He (Microsoft 북경 연구소)
- Residual block



  - ReLU를 통해 전달되는 비선형 경로와 레이어를 하나 혹은 두 단계씩 건너뛰는 skip connection(or shortcut)을 결합



- 이론적으로는 network가 깊어질수록 training set에 대해서는 error가 줄어들어야 하지만 training이 어려워지기 때문에 error가 상승할 수 있음
- Vanishing/exploding gradient problem 발생
- Skip connection을 사용함으로써 얕은 모델 보다 높은 training error를 가질 수 없음

Keras가 제공하는 ImageNet classification model

- 각 유명 모형과 학습된 weight를 제공 https://keras.io/applications/#usage-examples-for-image-classification-models (https://keras.io/applications/#usage-examples-for-image-classification-models)

In [5]:

```
from keras.applications.vgg16 import VGG16
import numpy as np

model = VGG16(weights='imagenet', include_top=False)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

References

- https://www.coursera.org/specializations/deep-learning (https://www.coursera.org/specializations/deep-learning)
- Hands on Machine Learning with Scikit-Learn and Tensorflow, Aurélien Géron (http://www.hanbit.co.kr/store/books/look.php?p_code=B9267655530)
- Deep Learning with Python, François Chollet, (https://www.manning.com/books/deep-learning-with-python)