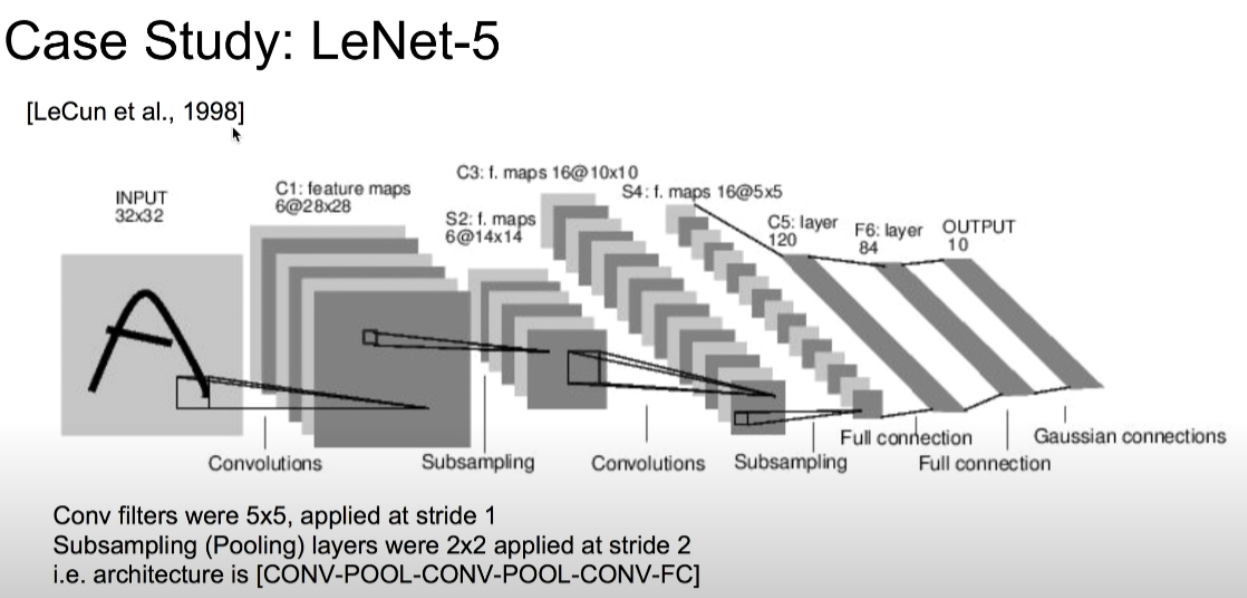
CNN Case Study

상태	CNN
담당자	

LeNet-5
AlexNet
GoogLeNet
ResNet
VGG

LeNet-5

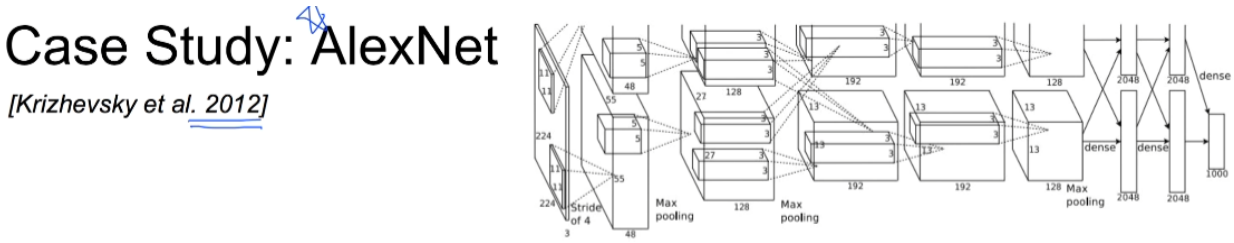


- Input : `1x32x32`
- Conv1 layer
  - C1 Convolution layer : `6x28x28`
    - 6개의 5x5 filter와 convolution 연산을 함으로써(output channel=6) 6개의 feature map을 얻을 수 있다.
  - S2 Pooling (subsampling) : `6x14x14`
    - kernel_size=2, stride=2로 pooling을 진행해 feature map의 크기를 절반으로(28 → 14) 줄인다.
- Conv2 layer
  - C3 Convolution layer : `16x10x10`

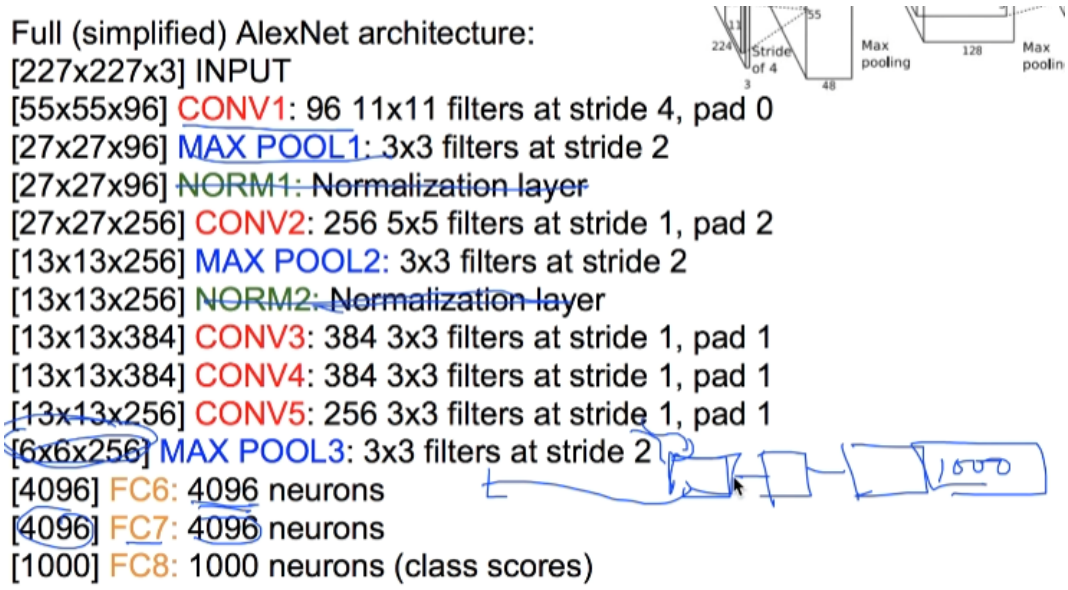
    왜 이미지의 크기가 10x10 인지는 다음의 링크를 참고하면 될 것 같다. (https://bskyvision.com/418) 다음 내용부터는 Sung Kim 교수님 강의에서 자세히 다루지 않고 넘어갔다.

    - S4 Pooling (subsampling) : `16x5x5`
- FC C5
- FC F6
    - Gaussian Connection

AlexNet

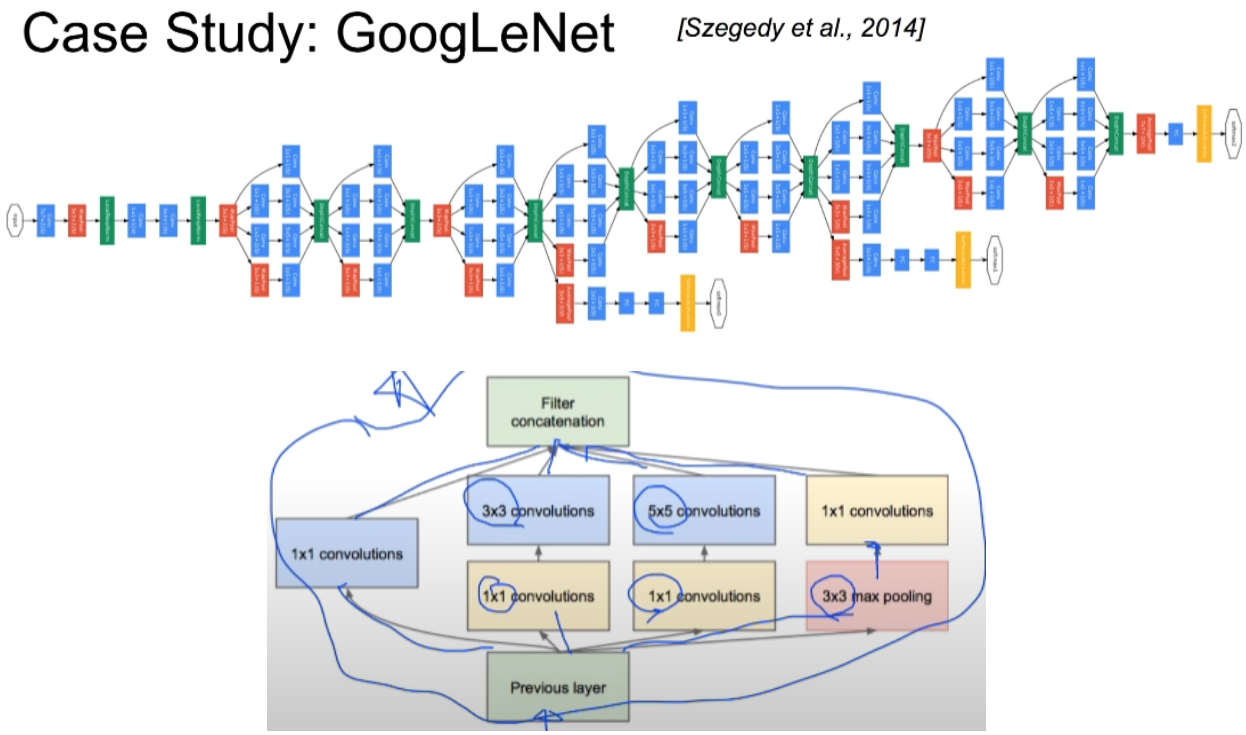


- Input : `3x227x227`
- Conv1 : `96x11x11` stride=4로 96개의 `3x11x11` 필터들이 convolution 연산되었다.
- Output : `96x55x55`



총 8개의 layer들을 사용했다. 마찬가지로 Sung Kim 교수님 강의에서는 AlexNet에 대한 자세한 내용을 다루진 않았다.

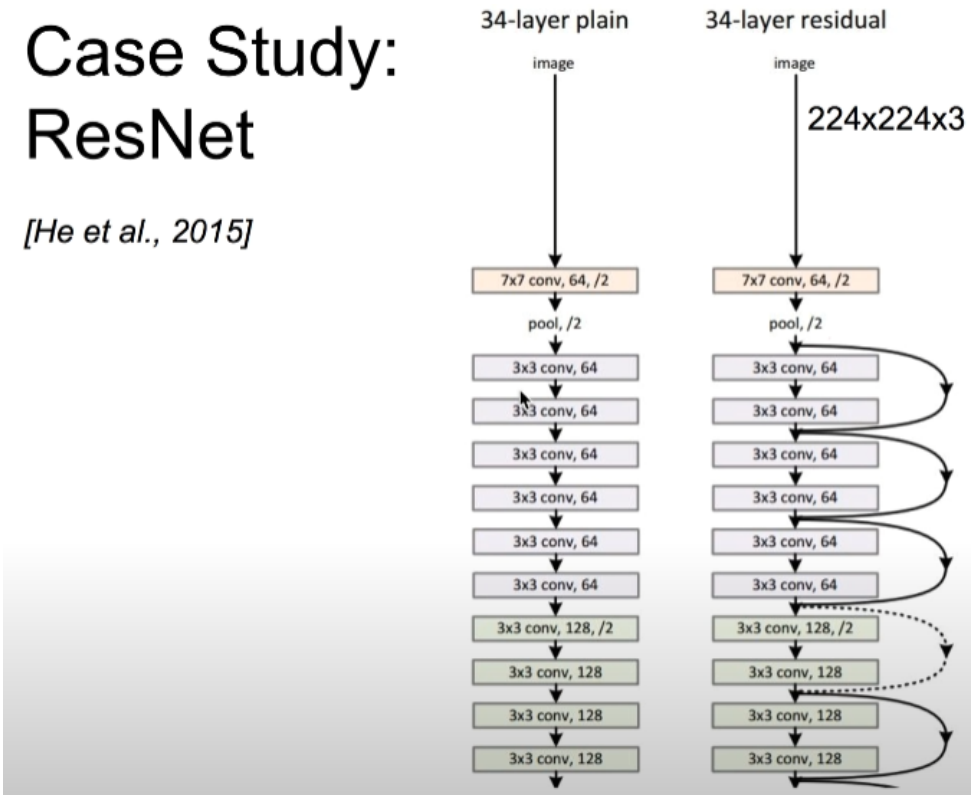
GoogLeNet



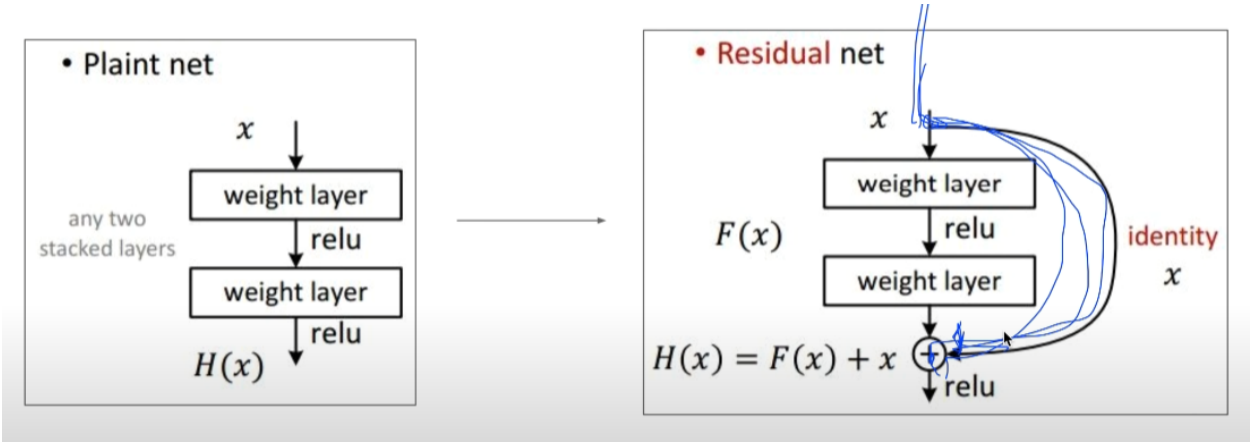
병렬적이면서도 한 개의 1x1 Conv은 따로 떨어져 나와 filter concatenation이 진행된다는 점이 인상깊은 구조이다.

ResNet

AlexNet이 8개의 layer들로 학습한 반면, ResNet은 152개의 layer들로 이루어져 있다. 🤔 그럼 학습이 어렵지 않을까?



일반적인 학습(왼쪽) 은 레이어가 단순히 sequential하게 연결되어 있다. 하지만 ResNet의 경우, 몇몇 레이어들이 레이어를 건너뛰어 연결되어 있다. (residual net)



따라서 레이어는 깊어 보여도 실제 학습할 때에는 그만큼 깊지 않다고 한다.

VGG

VGG 16

모든 레이어의 크기를 3x3으로 설정한 것이 특징이다.

```
class VGG(nn.Module):
    def __init__(self, features, num_classes=1000, init_weights=True):
        super(VGG, self).__init__()

        self.features = features #convolution

        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))

        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )#FC layer

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.features(x) #Convolution
        x = self.avgpool(x) # avgpool
        x = x.view(x.size(0), -1) #
        x = self.classifier(x) #FC layer
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)
```

여기서 `feature` 파라미터로 `make_layers(cfg['custom'])`이 사용되는데,

```
cfg = {
    'A': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'], #8 + 3 =11 == vgg11
    'B': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'], # 10 + 3 = vgg 13
    'D': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'], #13 + 3 = vgg 16
    'E': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'], # 16 +3 =vgg 19
```

```
    'custom' : [64,64,64,'M',128,128,128,'M',256,256,256,'M']
}
```

```
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3

    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v

    return nn.Sequential(*layers)
```

이다. `cfg['custom']`의 경우 `'custom' : [64,64,64,'M',128,128,128,'M',256,256,256,'M']` 이므로 for문을 돌면서 다음과 같은 convolution network을 만든다.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): ReLU(inplace)
  (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace)
  (9): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (10): ReLU(inplace)
  (11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (12): ReLU(inplace)
  (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace)
  (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): ReLU(inplace)
  (18): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (19): ReLU(inplace)
  (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```