VGG16 디자인

2.1 ARCHITECTURE

During training, the input to our ConvNets is a fixed-size 224×224 RGB image. The only pre-processing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

conv3. layer

kernel_size := 3×3
padding := 1
stride := 1

A stack of c
three Fully-
way ILSVR
the soft-ma

All hidden l
We note th
(LRN) norn
does not im
sumption a
of (Krizhev

```python
for x in architecture:
    if type(x) == int:
        """ 순회하면서 Conv 레이어를 쌓는 과정.
            64, 128, 256, 512 레이버 부분만 쌓음
        """
        out_channels = x  # 해당 레이어에서 출력하는 feature_map의 채널 길이

        layers += [ nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=(3,3), stride=(1,1), padding=(1,1)),
                    nn.BatchNorm2d(x),
                    nn.ReLU(),
                    ]

        in_channels = x  # 출력된 feature_map은 다음 레이어에서 입력으로 들어가니까

    elif x == 'M':
        """ MaxPooling
        feature_map 의 hight, width 사이즈만 줄어들지,
        채널 길이는 그대로 유지됨
        """
        layers += [nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))]
```

2.1 Architecture

```python
        """ Flatten and Linear layers 정의
        fully-connected layers
        """
        self.fcs = nn.Sequential(   nn.Linear(512*7*7, 4096),
                                    nn.ReLU(),
                                    nn.Dropout(p=0.5),
                                    nn.Linear(4096, 4096),
                                    nn.ReLU(),
                                    nn.Dropout(p=0.5),
                                    nn.Linear(4096, num_classes),
                                )
```

B image. The only pre-
ing set, from each pixel.
ve use filters with a very
n of left/right, up/down,
rs, which can be seen as
The convolution stride is
l resolution is preserved
pooling is carried out by
onv. layers are followed
th stride 2.

A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

VGG16

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 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 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 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

```python
VGG_types = {
    'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'VGG13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'],
    'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'],
}
```

```python
    def forward(self, x):
        x = self.conv_layers(x) # 입력 := [1, 3, 224, 224]  -> 출력 := [1, 512, 7, 7]
        x = x.reshape(x.shape[0], -1)   # for flatten
                                        # x.shape[0] 부분은 Batch_channel
        x = self.fcs(x)
        return x
```

After then,
flatten and '4096 × 4096 × num_classes' linear layers