InceptionV3

Kyeong Hwan Moon

01 Dataset

Inspection


n02097047 (196)


n01682714 (40)


n03134739 (522)


n04254777 (806)


n02859443 (449)


n02096177 (192)


n02107683 (239)

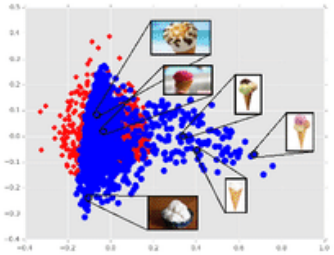

n01443537 (1)


n02264363 (318)


Icecream - ILSVRC12 (IN)


Icecream - WIN

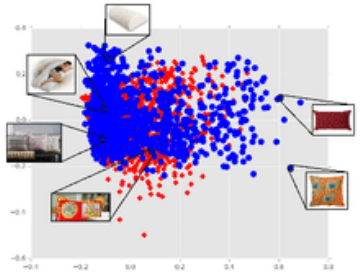

Icecream - WINC


ILSVR red, WINC blue


Pillow - ILSVRC12 (IN)

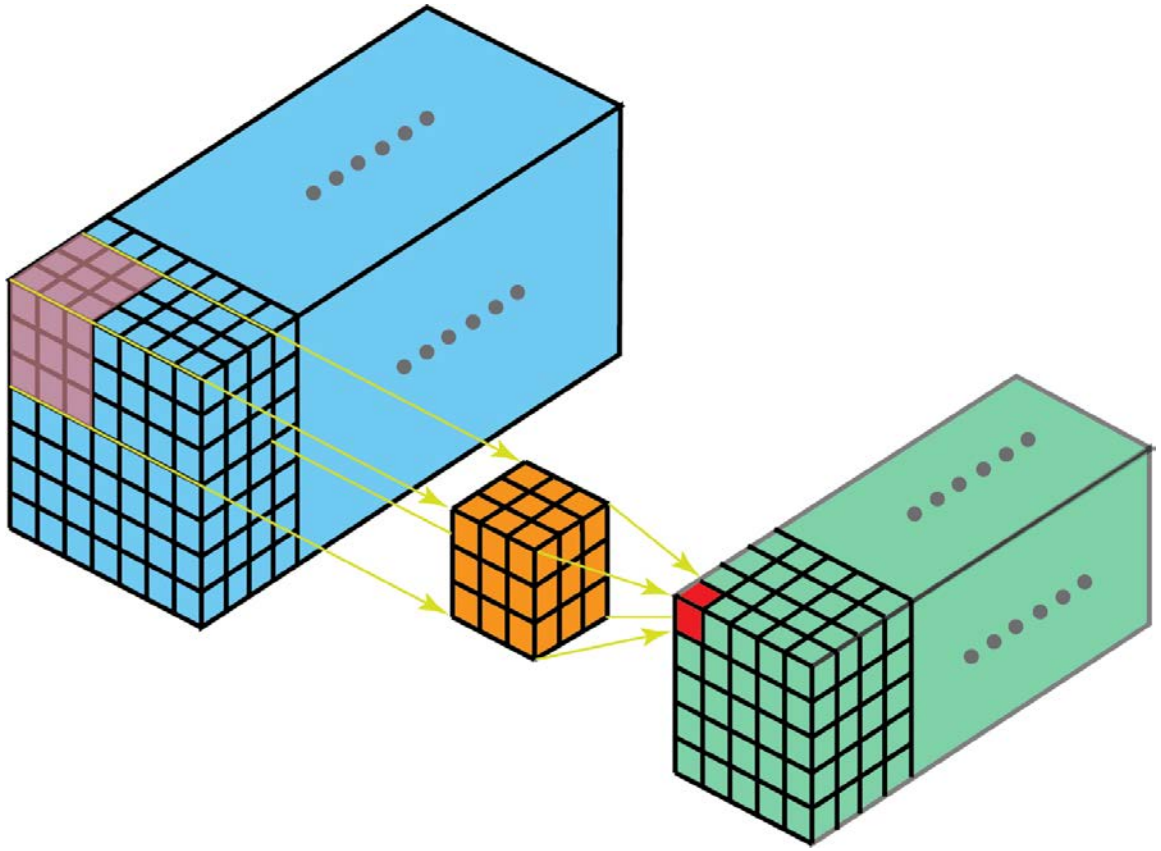

Pillow - WIN

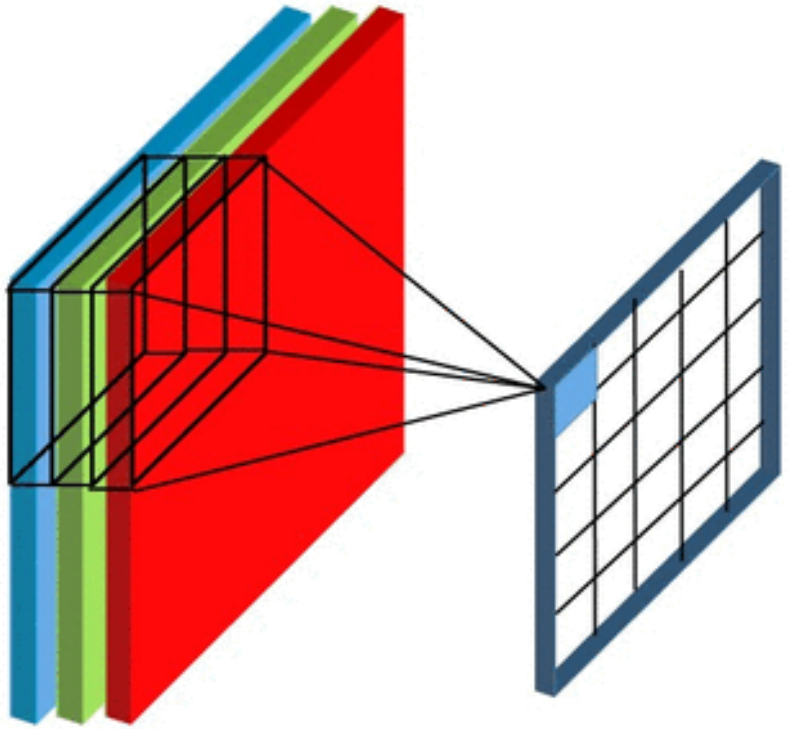

Pillow - WINC


ILSVR red, WINC blue

01 Inception Module

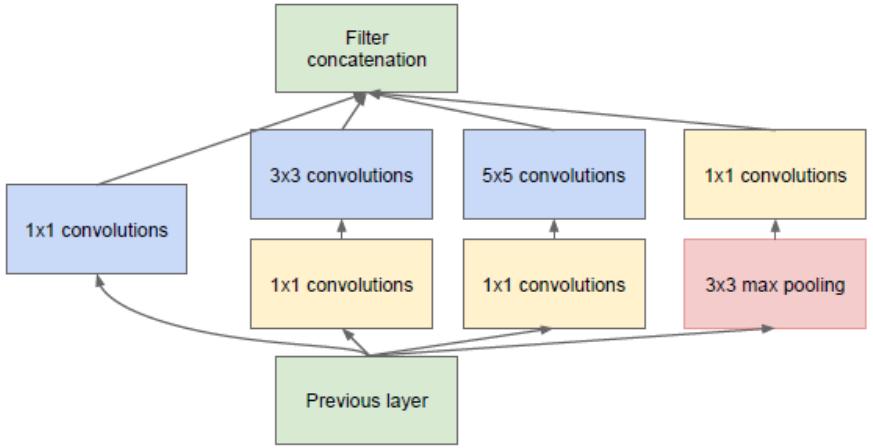
Naïve version



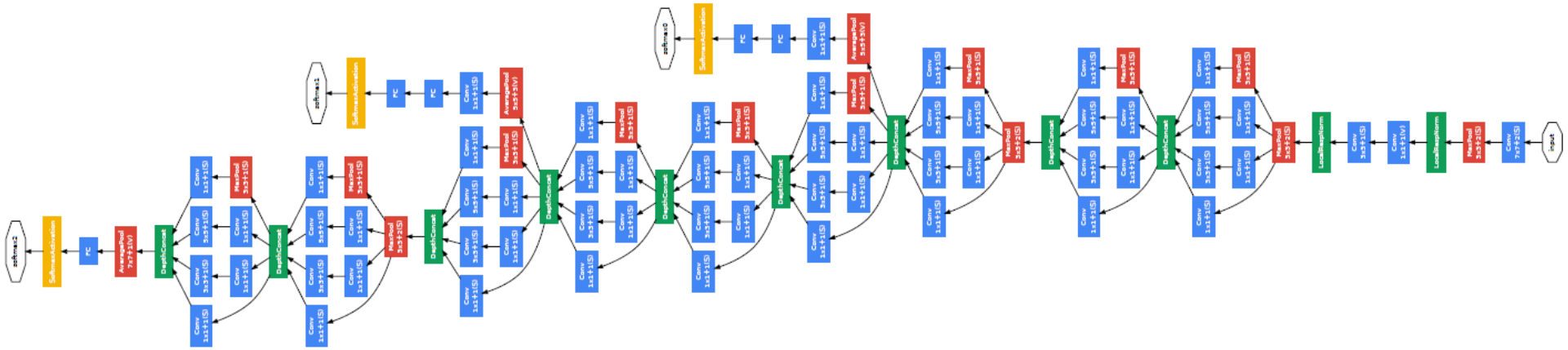


01 GoogLeNet

GoogleNet



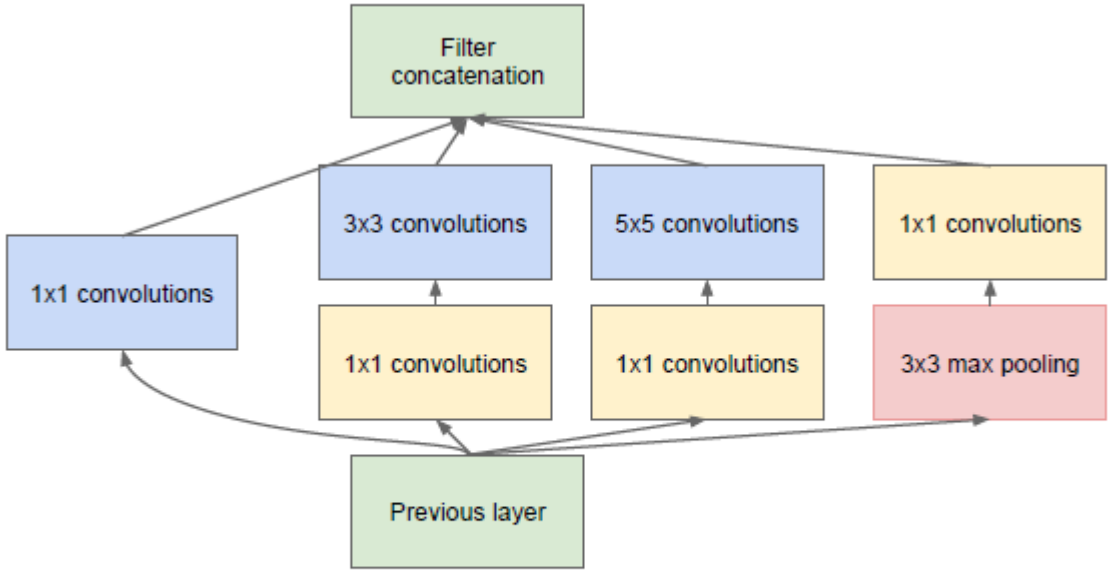
(b) Inception module with dimension reductions



- Large depth of the network
- Auxiliary classifiers are calculated while training
- Their loss gets added to the total loss of the network with a discount weight

01 Inception Module

Original



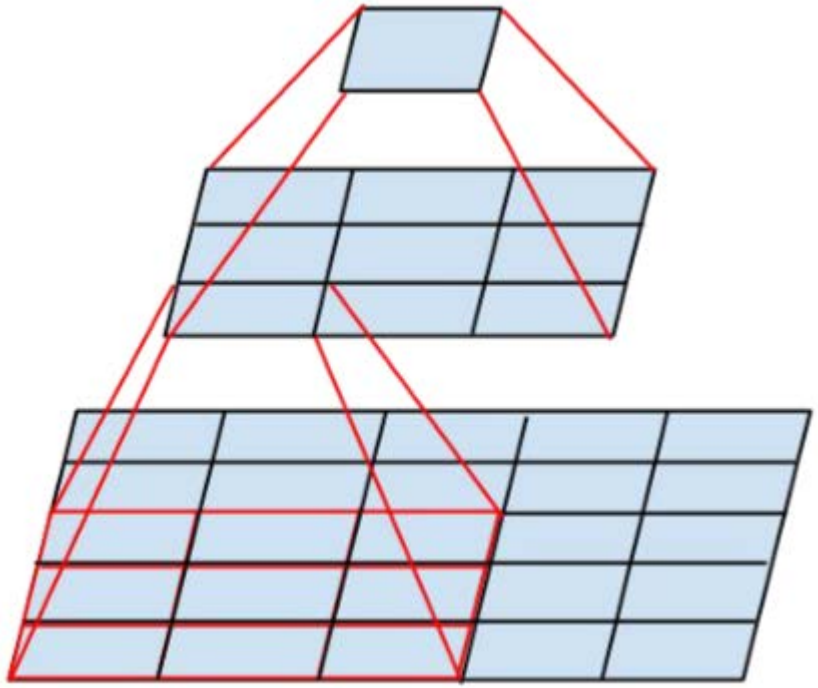
(b) Inception module with dimension reductions

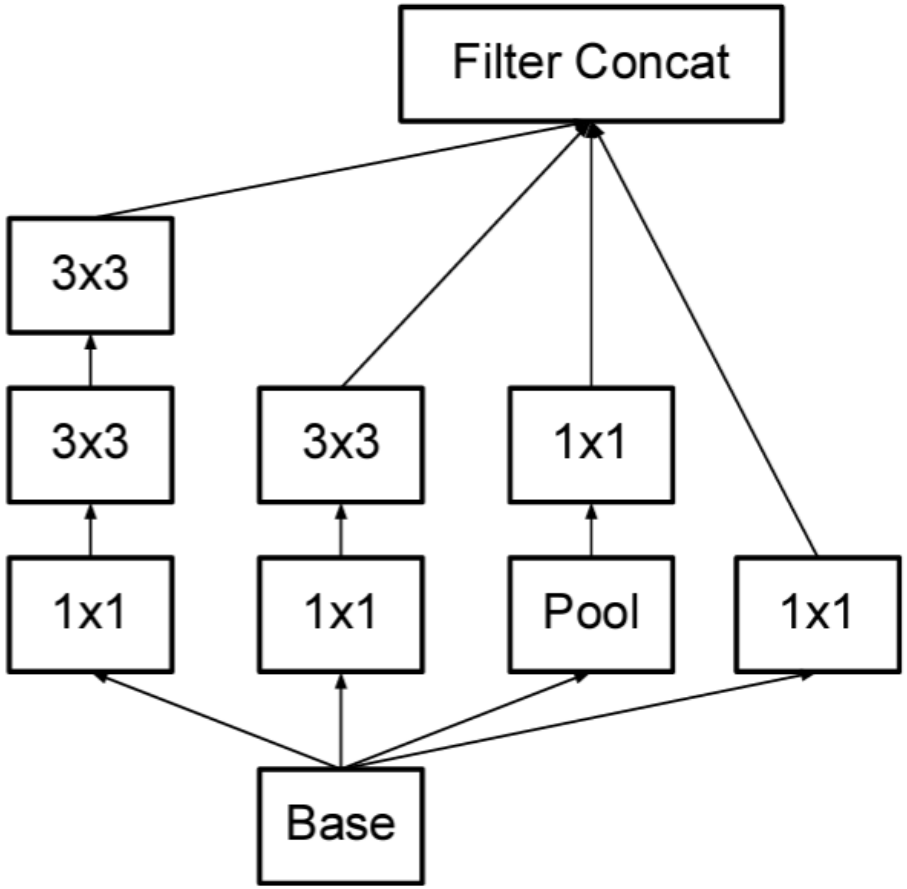
- Used 1x1 convolutional filter
- Calculation could be cheaper than naïve version

Factorization

01 Inception Block

5x5 to 3x3

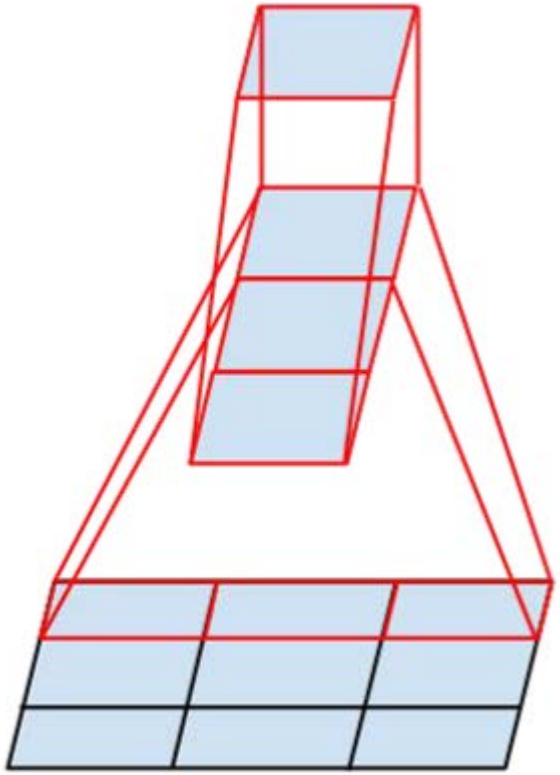


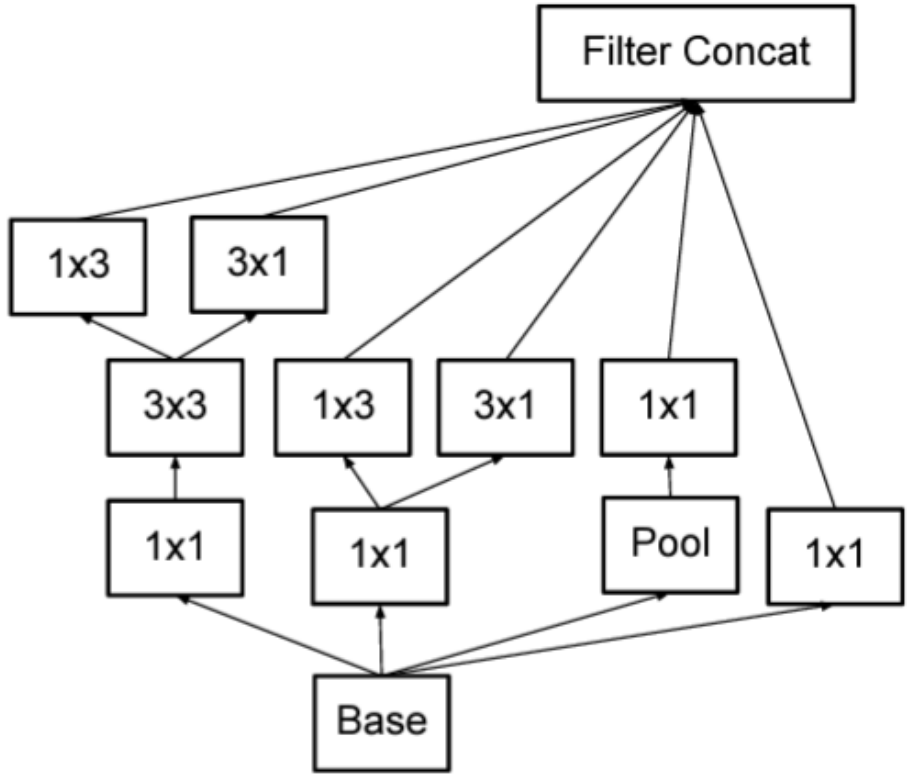


- 5 x 5 = 25
- 3 x 3 + 3 x 3 = 18

01 Inception Block

3x3 to 1x3 and 3x1

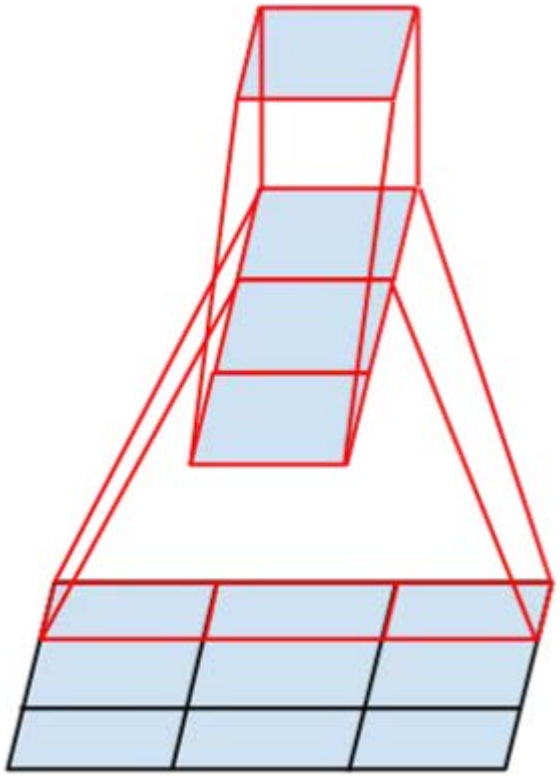


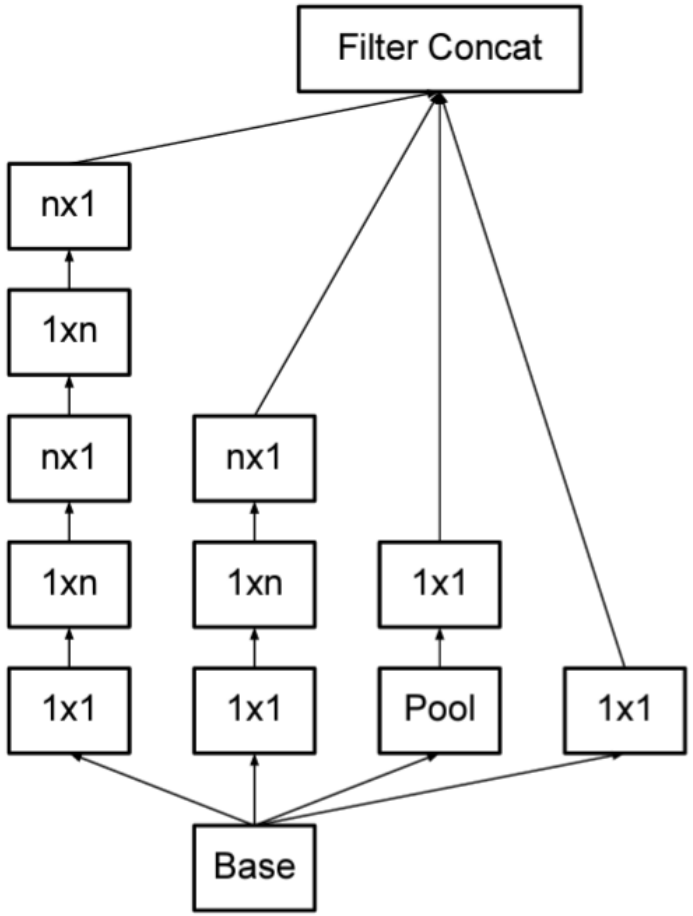


- 3 x 3 = 9
- 1 x 3 + 3 x 1 = 6

01 Inception Block

7x7 Factorization

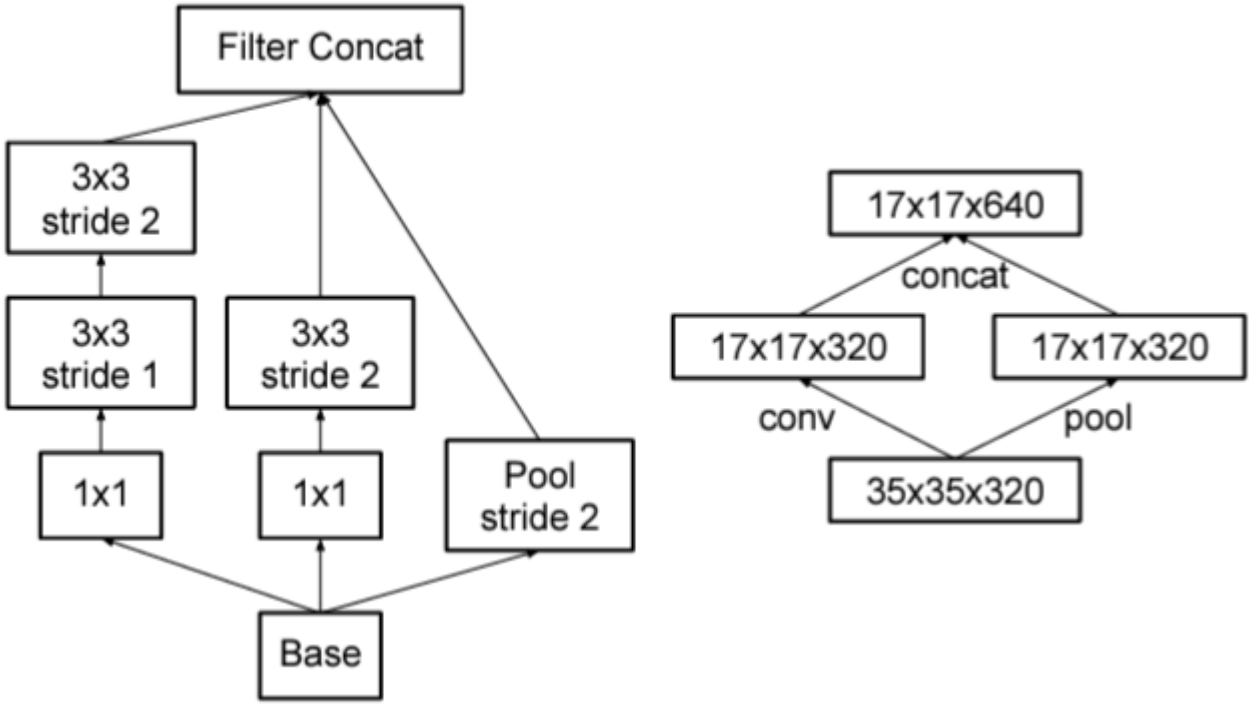




- n = 7

01 Reduction Block

Reduction



01 InceptionV3

Table

type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

01 Code

```python
class BasicConv2d(nn.Module):
  def __init__(self, in_channel, out_channel, **kwargs):
    super(BasicConv2d, self).__init__()
    self.conv = nn.Conv2d(in_channel, out_channel, bias=False, **kwargs)
    self.bn = nn.BatchNorm2d(out_channel, eps=0.001)

  def forward(self, x):
    x = self.conv(x)
    x = self.bn(x)
    return F.relu(x, inplace=True)
```

```python
class InceptionAux(nn.Module):
  def __init__(self, in_channel, num_classes, conv_block=None):
    super(InceptionAux, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d
    self.conv0 = conv_block(in_channel, 128, kernnel_size=1)
    self.conv1 = conv_block(128, 768, kernel_size=5)
    self.conv1.stddev = 0.01
    self.fc = nn.Linear(768, num_classes)
    self.fc.stddev = 0.001

  def forward(self, x):
    x = F.avg_pool2d(x, kernel_size=5, stride=3)
    x = self.conv0(x)
    x = self.conv1(x)
    x = F.adaptive_avg_pool2d(x, (1, 1))
    x = torch.flatten(x, 1)
    x = self.fc(x)
    return x
```

01 Code

```
class InceptionA(nn.Module):
  def __init__(self, in_channel, pool_features, conv_block=None):
    super(InceptionA, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d
    self.branch1x1 = conv_block(in_channel, 64, kernel_size=1)

    self.branch5x5_1 = conv_block(in_channel, 48, kernel_size=1)
    self.branch5x5_2 = conv_block(48, 64, kernel_size=5, padding=2)

    self.branch3x3_1 = conv_block(in_channel, 64, kernel_size=1)
    self.branch3x3_2 = conv_block(64, 96, kernel_size=3, padding=1)
    self.branch3x3_3 = conv_block(96, 96, kernel_size=3, padding=1)

    self.branch_pool = conv_block(in_channel, pool_features, kernel_size=1)

  def _forward(self, x):
    branch1x1 = self.branch1x1(x)

    branch5x5 = self.branch5x5_1(x)
    branch5x5 = self.branch5x5_2(branch5x5)

    branch3x3 = self.branch3x3_1(x)
    branch3x3 = self.branch3x3_2(branch3x3)
    branch3x3 = self.branch3x3_3(branch3x3)

    branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
    branch_pool = self.branch_pool(branch_pool)

    out = [branch1x1, branch5x5, branch3x3, branch_pool]
    return out
```

```
class InceptionB(nn.Module):
  def __init__(self, in_channel, conv_block=None):
    super(InceptionB, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d
    self.branch3x3 = conv_block(in_channel, 384, kernel_size=3, padding=2)

    self.branch3x3_1 = conv_block(in_channel, 64, kernel_size=1)
    self.branch3x3_2 = conv_block(64, 96, kernel_size=3, padding=1)
    self.branch3x3_3 = conv_block(96, 96, kernel_size=3, stride=2)

  def _forward(self, x):
    branch3x3 = self.branch3x3(x)

    branch3x3_ = self.branch3x3_1(x)
    branch3x3_ = self.branch3x3_2(branch3x3_)
    branch3x3_ = self.branch3x3_3(branch3x3_)

    branch_pool = F.avg_pool2d(x, kernel_size=3, stride=2)

    out = [branch3x3, branch3x3_, branch_pool]
    return out

  def forward(self, x):
    out = self._forward(x)
    return torch.cat(out, 1)
```

01 Code

```python
class InceptionC(nn.Module):
  def __init__(self, in_channel, channels_7x7, conv_block=None):
    super(InceptionC, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d

    self.branch1x1 = conv_block(in_channel, 192, kernel_size=1)

    c7 = channels_7x7
    self.branch7x7_1 = conv_block(in_channel, c7, kernel_size=1)
    self.branch7x7_2 = conv_block(c7, c7, kernel_size=(1, 7), padding=(0, 3))
    self.branch7x7_3 = conv_block(c7, 192, kernel_size=(7, 1), padding=(3, 0))

    self.branch7x7dbl_1 = conv_block(in_channel, c7, kernel_size=1)
    self.branch7x7dbl_2 = conv_block(c7, c7, kernel_size=(7, 1), padding=(3, 0))
    self.branch7x7dbl_3 = conv_block(c7, c7, kernel_size=(1, 7), padding=(0, 3))
    self.branch7x7dbl_4 = conv_block(c7, c7, kernel_size=(7, 1), padding=(3, 0))
    self.branch7x7dbl_5 = conv_block(c7, 192, kernel_size=(1, 7), padding=(0, 3))

    self.branch_pool = self.conv_block(in_channel, 192, kernel_size=1)
```

```python
def _forward(self, x):
  branch1x1 = self.branch1x1(x)

  branch7x7 = self.branch7x7_1(x)
  branch7x7 = self.branch7x7_2(branch7x7)
  branch7x7 = self.branch7x7_3(branch7x7)

  branch7x7dbl = self.branch7x7dbl_1(x)
  branch7x7dbl = self.branch7x7dbl_2(branch7x7dbl)
  branch7x7dbl = self.branch7x7dbl_3(branch7x7dbl)
  branch7x7dbl = self.branch7x7dbl_4(branch7x7dbl)
  branch7x7dbl = self.branch7x7dbl_5(branch7x7dbl)

  branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
  branch_pool = self.branch_pool(x)

  out = [branch1x1, branch7x7, branch7x7dbl, branch_pool]
  return out

def forward(self, x):
  out = self._forward(x)
  return torch.cat(out, 1)
```

01 Code

```
class InceptionD(nn.Module):
  def __init__(self, in_channel, conv_block=None):
    super(InceptionD, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d
    self.branch3x3_1 = conv_block(in_channel, 192, kernel_size=1)
    self.branch3x3_2 = conv_block(192, 320, kernel_size=3, stride=2)

    self.branch7x7x3_1 = conv_block(in_channel, 192, kernel_size=1)
    self.branch7x7x3_2 = conv_block(192, 192, kernel_size=(1, 7), padding=(0, 3))
    self.branch7x7x3_3 = conv_block(192, 192, kernel_size=(7, 1), padding=(3, 0))
    self.branch7x7x3_4 = conv_block(192, 192, kernel_size=3, stride=2)

  def _forward(self, x):
    branch3x3 = self.branch3x3_1(x)
    branch3x3 = self.branch3x3_2(branch3x3)

    branch7x7x3 = self.branch7x7x3_1(x)
    branch7x7x3 = self.branch7x7x3_2(branch7x7x3)
    branch7x7x3 = self.branch7x7x3_3(branch7x7x3)
    branch7x7x3 = self.branch7x7x3_4(branch7x7x3)

    branch_pool = F.max_pool2d(x, kernel_size=3, stride=2)
    out = [branch3x3, branch7x7x3, branch_pool]
    return out

  def forward(self, x):
    out = self._forward(x)
    return torch.cat(out, 1)
```

01 Code

```python
class InceptionE(nn.Module):
  def __init__(self, in_channel, conv_block=None):
    super(InceptionE, self).__init__()
    if conv_block is None:
      conv_block = BasicConv2d
    self.branch1x1 = conv_block(in_channel, 320, kernel_size=1)

    self.branch3x3_1 = conv_block(in_channel, 384, kernel_size=1)
    self.branch3x3_2a = conv_block(384, 384, kernel_size=(1, 3), padding=(0, 1))
    self.branch3x3_2b = conv_block(384, 384, kernel_size=(3, 1), padding=(1, 0))

    self.branch3x3dbl_1 = conv_block(in_channel, 448, kernel_size=1)
    self.branch3x3dbl_2 = conv_block(448, 384, kernel_size=3, padding=1)
    self.branch3x3dbl_3a = conv_block(384, 384, kernel_size=(1, 3), padding=(0, 1))
    self.branch3x3dbl_3b = conv_block(384, 384, kernel_size=(3, 1), padding=(1, 0))

    self.branch_pool = conv_block(in_channel, 192, kernel_size=1)
```

```python
def _forward(self, x):
  branch1x1 = self.branch1x1(x)

  branch3X3 = self.branch3x3_1(x)
  branch3x3 = [
      self.branch3x3_2a(branch3x3),
      self.branch3x3_2b(branch3x3)
  ]
  branch3x3 = torch.cat(branch3x3, 1)

  branch3x3dbl = self.branch3x3dbl_1(x)
  branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
  branch3x3dbl = [
      self.branch3x3dbl_3a(branch3x3dbl),
      self.branch3x3dbl_3b(branch3x3dbl)
  ]
  branch3x3dbl = torch.cat(branch3x3dbl, 1)

  branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
  branch_pool = self.branch_pool(branch_pool)

  out = [branch1x1, branch3x3, branch3x3dbl, branch_pool]
  return out

def forward(self, x):
  out = self._forward(x)
  return torch.cat(out, 1)
```

01 Code

```
class InceptionV3(nn.Module):
  def __init__(self, num_classes=1000, aux_logits=True, transform_input=False,
               inception_blocks=None, init_weights=True):
    super(InceptionV3, self).__init__()
    if inception_blocks is None:
      inception_blocks = [
          BasicConv2d, InceptionA, InceptionB,InceptionC,
          InceptionD, InceptionE, InceptionAux
      ]

    conv_block = inception_blocks[0]
    inception_a = inception_blocks[1]
    inception_b = inception_blocks[2]
    inception_c = inception_blocks[3]
    inception_d = inception_blocks[4]
    inception_e = inception_blocks[5]
    inception_aux = inception_blocks[6]

    self.aux_logits = aux_logits
    self.transform_input = transform_input
    self.Conv2d_1a_3x3 = conv_block(3, 32, kernel_size=3, stride=2)
    self.Conv2d_2a_3x3 = conv_block(32, 32, kernel_size=3)
    self.Conv2d_2b_3x3 = conv_block(32, 64, kernel_size=3, padding=1)
    self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.Conv2d_3b_1x1 = conv_block(64, 80, kernel_size=1)
    self.Conv2d_4a_3x3 = conv_block(80, 192, kernel_size=3)
    self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.Mixed_5b = inception_a(192, pool_features=32)
    self.Mixed_5c = inception_a(256, pool_features=64)
    self.Mixed_5d = inception_a(288, pool_features=64)
    self.Mixed_6a = inception_b(288)
    self.Mixed_6b = inception_c(768, channels_7x7=128)
    self.Mixed_6c = inception_c(768, channels_7x7=160)
    self.Mixed_6d = inception_c(768, channels_7x7=160)
    self.Mixed_6e = inception_c(768, channels_7x7=192)
    if aux_logits:
        self.AuxLogits = inception_aux(768, num_classes)
    self.Mixed_7a = inception_d(768)
    self.Mixed_7b = inception_e(1280)
    self.Mixed_7c = inception_e(2048)
    self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
    self.dropout = nn.Dropout()
    self.fc = nn.Linear(2048, num_classes)

    if init_weights:
      for m in self.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
          import scipy.stats as stats
          stddev = m.stddev if hasattr(m, 'stddev') else 0.1
          X = stats.truncnorm(-2, 2, scale=stddev)
          values = torch.as_tensor(X.rvs(m.weight.numel()), dtype=m.weight.dtype)
          values = values.view(m.weight.size())
          with torch.no_grad():
            m.weight.copy_(values)
        elif isinstance(m, nn.BatchNorm2d):
          nn.init.constant_(m.weight, 1)
          nn.init.constant_(m.bias, 0)

  def _transform_input(self, x):
    if self.transform_input:
      x_ch0 = torch.unsqueeze(x[:, 0], 1) * (0.229 / 0.5) + (0.485 - 0.5) / 0.5
      x_ch1 = torch.unsqueeze(x[:, 1], 1) * (0.224 / 0.5) + (0.456 - 0.5) / 0.5
      x_ch2 = torch.unsqueeze(x[:, 2], 1) * (0.225 / 0.5) + (0.406 - 0.5) / 0.5
```

01 Code

```
def _transform_input(self, x):
  if self.transform_input:
    x_ch0 = torch.unsqueeze(x[:, 0], 1) * (0.229 / 0.5) + (0.485 - 0.5) / 0.5
    x_ch1 = torch.unsqueeze(x[:, 1], 1) * (0.224 / 0.5) + (0.456 - 0.5) / 0.5
    x_ch2 = torch.unsqueeze(x[:, 2], 1) * (0.225 / 0.5) + (0.406 - 0.5) / 0.5
    x = torch.cat((x_ch0, x_ch1, x_ch2), 1)
  return x

def _forward(self, x):
  x = self.Conv2d_1a_3x3(x)
  x = self.Conv2d_2a_3x3(x)
  x = self.Conv2d_2b_3x3(x)
  x = self.maxpool1(x)
  x = self.Conv2d_3b_1x1(x)
  x = self.Conv2d_4a_3x3(x)
  x = self.maxpool2(x)
  x = self.Mixed_5b(x)
  x = self.Mixed_5c(x)
  x = self.Mixed_5d(x)
  x = self.Mixed_6a(x)
  x = self.Mixed_6b(x)
  x = self.Mixed_6c(x)
  x = self.Mixed_6d(x)
  x = self.Mixed_6e(x)
  aux_defined = self.training and self.aux_logits
  if aux_defined:
    aux = self.AuxLogits(x)
  else:
    aux = None
```

```
    x = self.Mixed_7a(x)
    x = self.Mixed_7b(x)
    x = self.Mixed_7c(x)
    x = self.avgpool(x)
    x = self.dropout(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)
    return x, aux

  def forward(self, x):
    out = self._forward(x)
    return torch.cat(out, 1)
```

Thank You