Computer Vision - ShuffleNet v2

Prof. Dr.Eng FITRI UTAMININGRUM, ST., MT

brone.ub.ac.id

ShuffleNet

ShuffleNet merupakan convolutional neural network yang didesain untuk perangkat mobile dengan kemampuan komputasi terbatas. Model ShuffleNetV2 dirancang agar akurat dan lebih cepat dijalankan pada perangkat GPU dan ARM.

- Model ShuffleNet dikenalkan oleh **Zhang et al (2017)** dalam paper "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices"
- **Ma et al. (2018)** mengajukan ShuffleNetV2 yang terinspirasi dari ShuffleNet dalam papernya "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design".

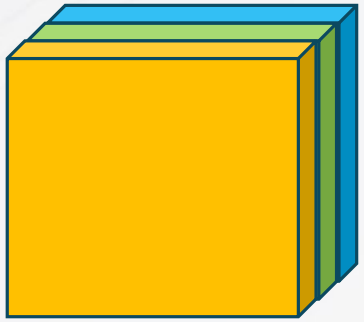
Kelebihan ShuffleNetV2

Berdasarkan hasil penelitian dari Ma et al. (2018), ShuffleNetV2 menunjukkan hasil yang lebih akurat dibanding ShuffleNetV1 dan dapat memproses lebih banyak gambar per detik.

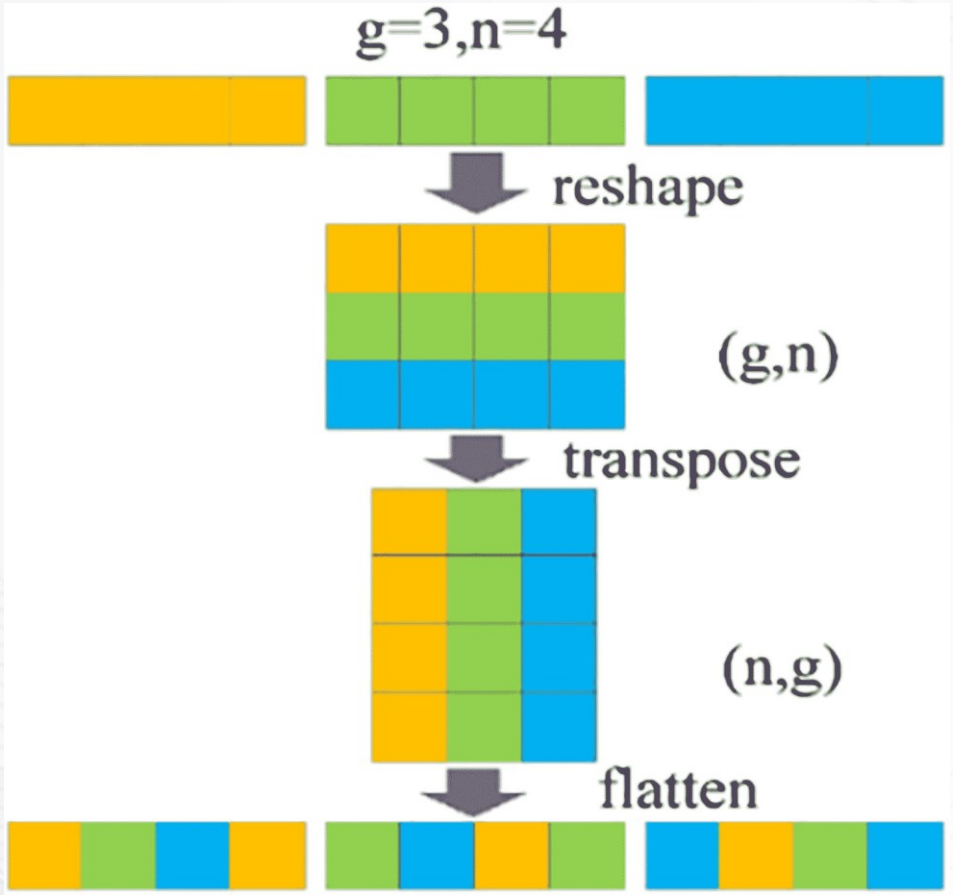
Model	mmAP(%)				GPU Speed (Images/sec.)			
FLOPs	40M	140M	300M	500M	40M	140M	300M	500M
Xception	21.9	29.0	31.3	32.9	178	131	101	83
ShuffleNet v1	20.9	27.0	29.9	32.9	152	85	76	60
MobileNet v2	20.7	24.4	30.0	30.6	146	111	94	72
ShuffleNet v2 (ours)	22.5	29.0	31.8	33.3	**188**	**146**	**109**	**87**
ShuffleNet v2* (ours)	**23.7**	**29.6**	**32.2**	**34.2**	183	138	105	83

Table 7: Performance on COCO object detection. The input image size is 800 × 1200. *FLOPs* row lists the complexity levels at 224 × 224 input size. For GPU speed evaluation, the batch size is 4. We do not test ARM because the *PSRoI Pooling* operation needed in [34] is unavailable on ARM currently.

Cara Kerja Channel Shuffle



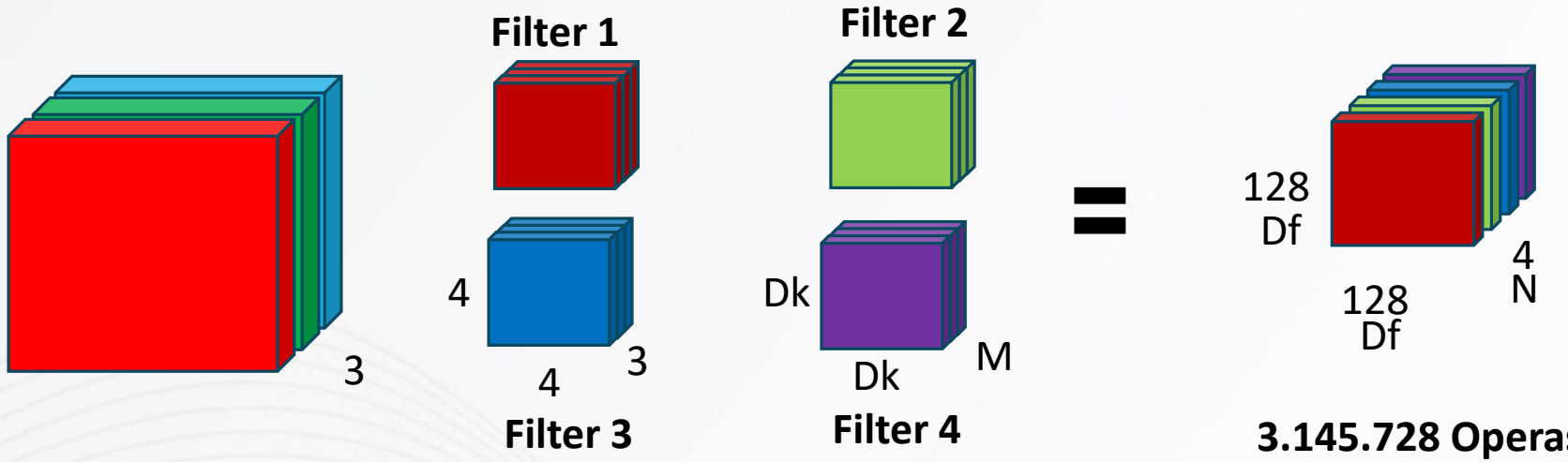
- Unit utama pada ShuflleNetV2 adalah unit Channel Shuffle.
- Channel Shuffle merupakan operasi yang membuat adanya komunikasi informasi antara channel yang berbeda. Pertukaran informasi ini dipercaya dapat meningkatkan akurasi dengan biaya komputasi yang rendah



- Pertukaran informasi ini dipercaya dapat meningkatkan akurasi dengan biaya komputasi yang rendah

Jumlah Operasi Konvolusi Biasa

ShuffleNetV2 memanfaatkan konvolusi Depthwise dan Pointwise. Kedua metode konvolusi ini memiliki jumlah operasi yang lebih sedikit, namun jumlah hasil yang dihasilkan sama dengan **konvolusi biasa**



Kompleksitas Operasi: Dk · Dk · M · N · Df · Df

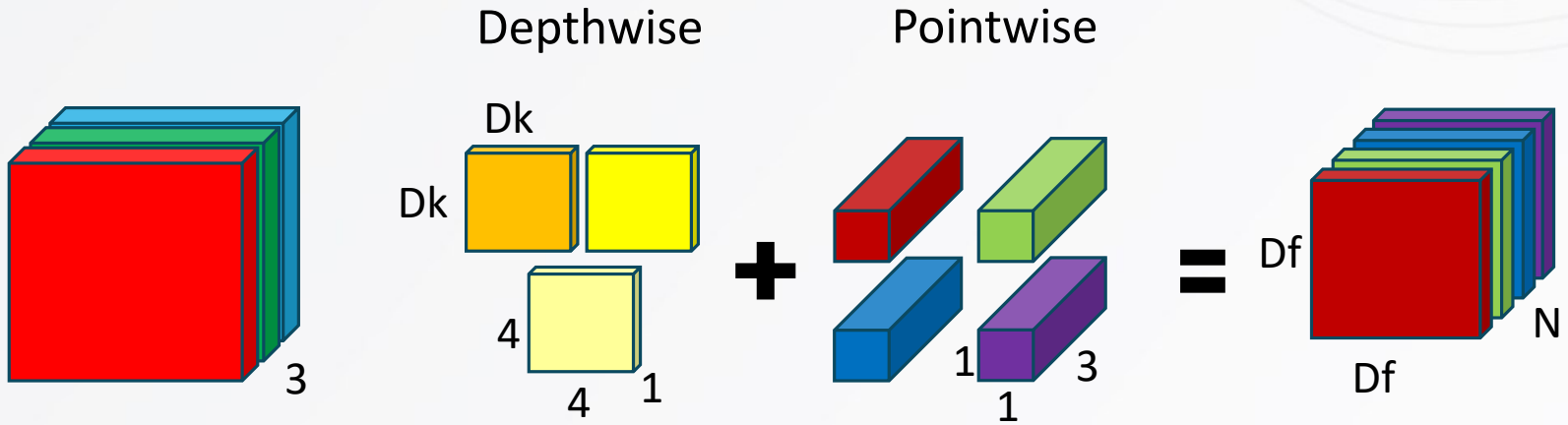
Dk = Dimensi (panjang dan lebar) filter = 4

M = Kedalaman filter (menyesuaikan kedalaman input) = 3

N = Jumlah filter = 4

Df = Dimensi output = 128

Jumlah Operasi Depthwise & Pointwise



Jumlah filter Depthwise mengikuti nilai M

786.432 + 196.608 = 983.040 operations

Kompleksitas Operasi: $Dk \cdot Dk \cdot M \cdot Df \cdot Df + M \cdot N \cdot Df \cdot Df$

Operasi menjadi lebih ringan dengan hasil yang sama

Dk = 4

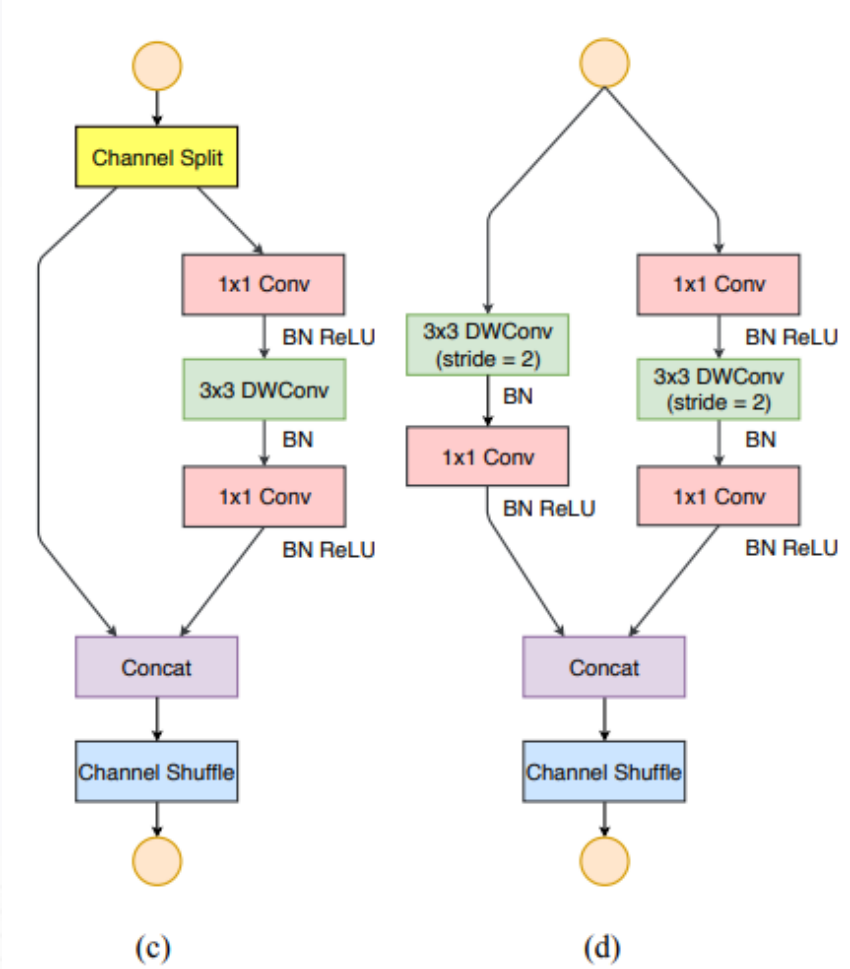
Df = 128

N = 4

M = 3

ShuffleNetV2 Basic Unit

- Berikut ini adalah blok utama dalam ShuffleNetV2
- Gambar (c) adalah blok yang digunakan ketika nilai stride 1
- Gambar (d) adalah blok yang digunakan ketika nilai stride 2
- 1x1 Conv = Pointwise Conv
- 3x3 DW Conv = Depthwise Conv



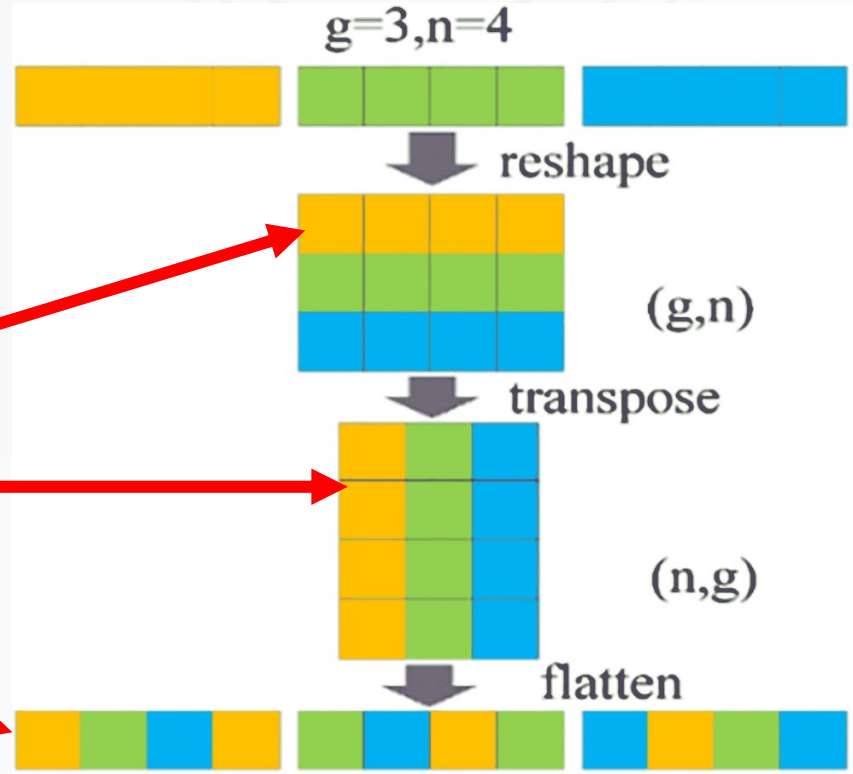
Arsitektur ShuffleNetV2

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Table 5: Overall architecture of ShuffleNet v2, for four different levels of complexities.

Channel Shuffle

```python
# ShuffleNetV2
def channel_shuffle(x: Tensor, groups: int) -> Tensor:
    batchsize, num_channels, height, width = x.size()
    channels_per_group = num_channels // groups

    # reshape
    x = x.view(batchsize, groups,
               channels_per_group, height, width)

    x = torch.transpose(x, 1, 2).contiguous()

    # flatten
    x = x.view(batchsize, -1, height, width)

    return x
```

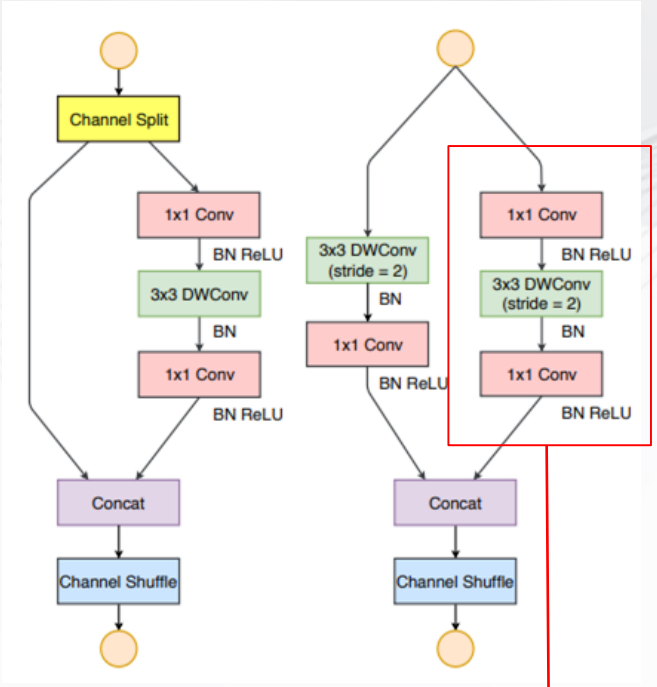


ShuffleNetV2 Basic Unit

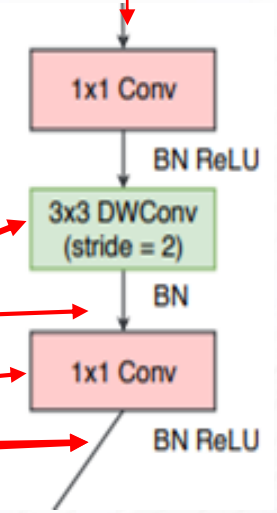
```python
class ShuffleNetV2_InvertedResidual(nn.Module):
    def __init__(
        self,
        inp: int,
        oup: int,
        stride: int
    ) -> None:
        super(ShuffleNetV2_InvertedResidual, self).__init__()

        if not (1 <= stride <= 3):
            raise ValueError('illegal stride value')
        self.stride = stride

        branch_features = oup // 2
        assert (self.stride != 1) or (inp == branch_features << 1)

        if self.stride > 1:
            self.branch1 = nn.Sequential(
                self.depthwise_conv(inp, inp, kernel_size=3, stride=self.stride, padding=1),
                nn.BatchNorm2d(inp),
                nn.Conv2d(inp, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
                nn.BatchNorm2d(branch_features),
                nn.ReLU(inplace=True),
            )
        else:
            self.branch1 = nn.Sequential()
        self.branch2 = nn.Sequential(
            nn.Conv2d(inp if (self.stride > 1) else branch_features,
                      branch_features, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(branch_features),
            nn.ReLU(inplace=True),
            self.depthwise_conv(branch_features, branch_features, kernel_size=3, stride=self.stride, padding=1),
            nn.BatchNorm2d(branch_features),
            nn.Conv2d(branch_features, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(branch_features),
            nn.ReLU(inplace=True),
        )
```

```python
    @staticmethod
    def depthwise_conv(
        i: int,
        o: int,
        kernel_size: int,
        stride: int = 1,
        padding: int = 0,
        bias: bool = False
    ) -> nn.Conv2d:
        return nn.Conv2d(i, o, kernel_size, stride, padding, bias=bias, groups=i)

    def forward(self, x: Tensor) -> Tensor:
        if self.stride == 1:
            x1, x2 = x.chunk(2, dim=1)
            out = torch.cat((x1, self.branch2(x2)), dim=1)
        else:
            out = torch.cat((self.branch1(x), self.branch2(x)), dim=1)

        out = channel_shuffle(out, 2)

        return out
```

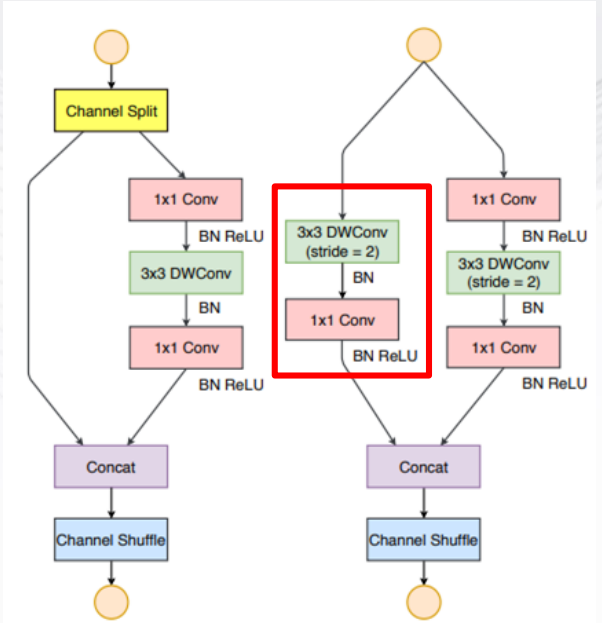
ShuffleNetV2 Basic Unit (Branch Kanan)



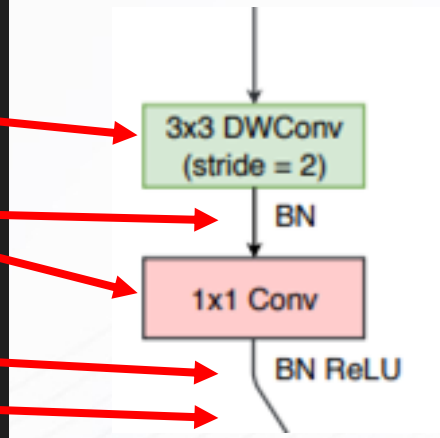

```
self.branch2 = nn.Sequential(
    nn.Conv2d(inp if (self.stride > 1) else branch_features,
              branch_features, kernel_size=1, stride=1, padding=0, bias=False),
    nn.BatchNorm2d(branch_features),
    nn.ReLU(inplace=True),
    self.depthwise_conv(branch_features, branch_features, kernel_size=3, stride=self.stride, padding=1),
    nn.BatchNorm2d(branch_features),
    nn.Conv2d(branch_features, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
    nn.BatchNorm2d(branch_features),
    nn.ReLU(inplace=True),
)
```



ShuffleNetV2 Basic Unit (Branch Kiri)

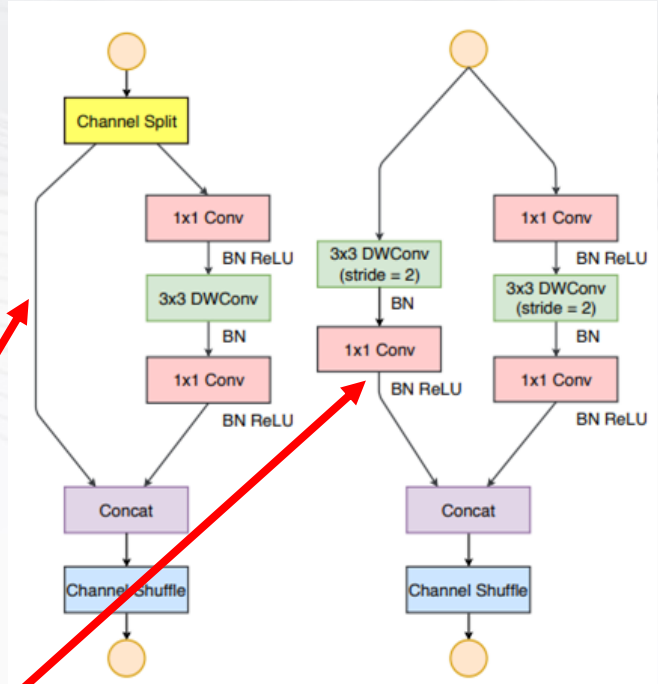


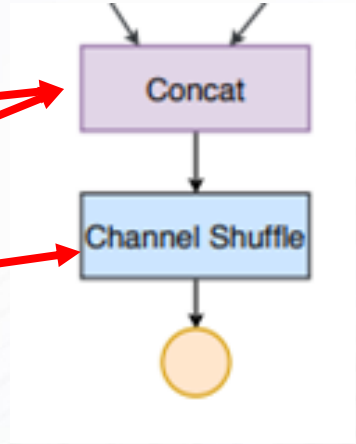

```python
if self.stride > 1:
    self.branch1 = nn.Sequential(
        self.depthwise_conv(inp, inp, kernel_size=3, stride=self.stride, padding=1),
        nn.BatchNorm2d(inp),
        nn.Conv2d(inp, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
        nn.BatchNorm2d(branch_features),
        nn.ReLU(inplace=True),
    )
else:
    self.branch1 = nn.Sequential()
```



ShuffleNetV2 Basic Unit (Concat dan Channel Shuffle)

```python
def forward(self, x: Tensor) -> Tensor:
    if self.stride == 1:
        x1, x2 = x.chunk(2, dim=1)
        out = torch.cat((x1, self.branch2(x2)), dim=1)
    else:
        out = torch.cat((self.branch1(x), self.branch2(x)), dim=1)

    out = channel_shuffle(out, 2)

    return out
```





ShuffleNet 1 dan 2

Fitur	ShuffleNet V1	ShuffleNet V2	Implikasi
Unit Dasar	Shuffle Unit	Channel Shuffle dan Channel Split	**Efisiensi:** V2 lebih fleksibel dalam mengatur aliran data antar channel.
Operasi Pointwise Convolution	Menggunakan depthwise separable convolution	Menggunakan group convolution dan channel shuffle	**Efisiensi:** V2 lebih efisien dalam mengurangi parameter dan komputasi.
Pengurangan Parameter	Menggunakan channel shuffle untuk mengurangi parameter	Menggunakan channel shuffle dan group convolution untuk mengurangi parameter lebih lanjut	**Ukuran Model:** Model V2 lebih kecil dan lebih ringan.
Akurasi	Cukup baik untuk tugas klasifikasi	Lebih tinggi, terutama untuk tugas deteksi objek	**Performa:** V2 memberikan akurasi yang lebih baik, terutama pada perangkat dengan daya komputasi terbatas.
Kompleksitas Arsitektur	Relatif sederhana	Lebih kompleks	**Implementasi:** V2 membutuhkan desain yang lebih cermat, tetapi memberikan fleksibilitas yang lebih tinggi.
Fokus Utama	Efisiensi komputasi	Keseimbangan antara efisiensi dan akurasi	**Tujuan:** V1 lebih fokus pada perangkat mobile dengan daya komputasi terbatas, sedangkan V2 lebih fleksibel untuk berbagai aplikasi.

TERIMAKASIH