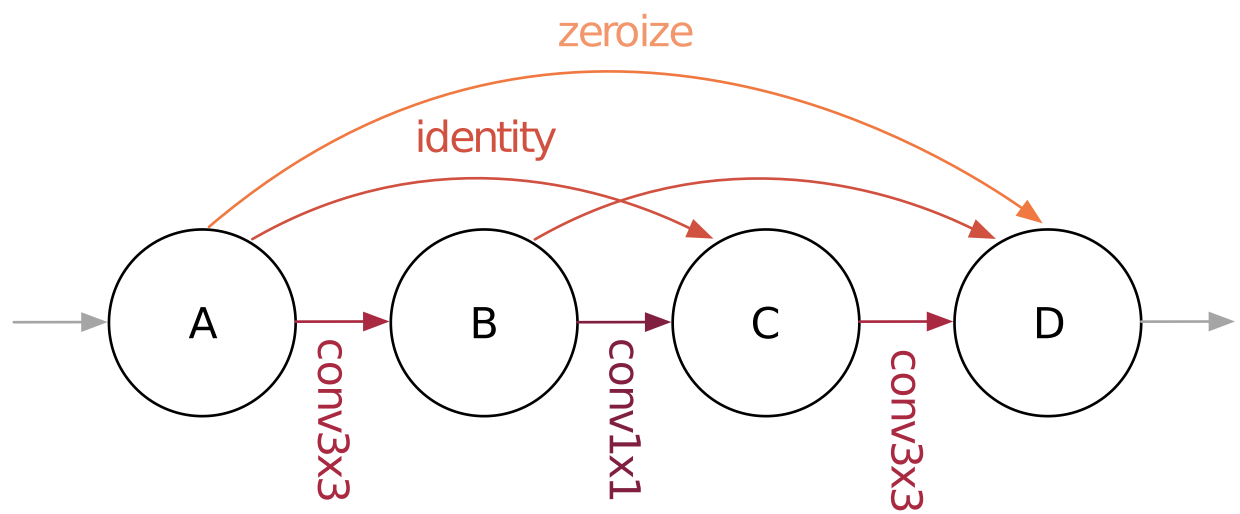
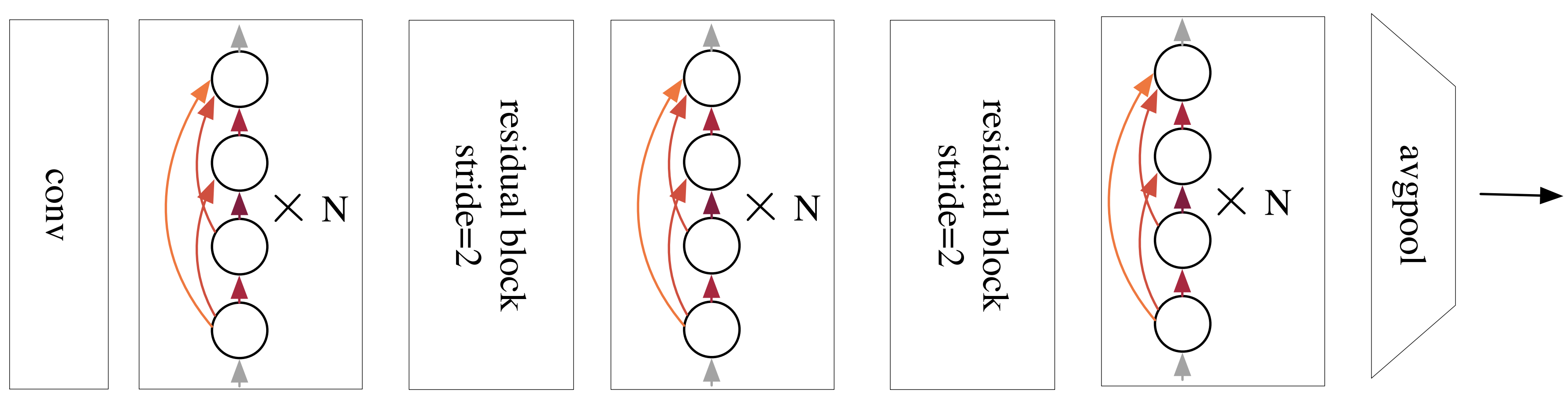
NAS is too coarse-grained

1

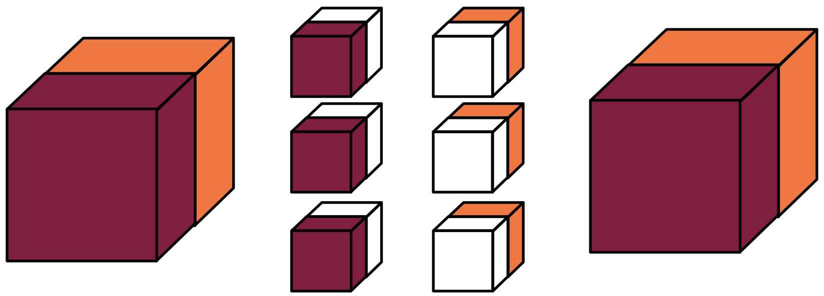


2



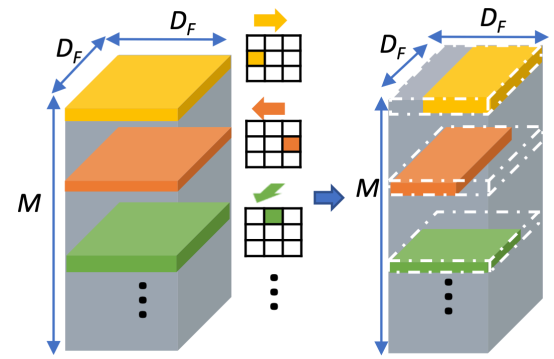
What do we really want from NAS?

Grouped convolutions



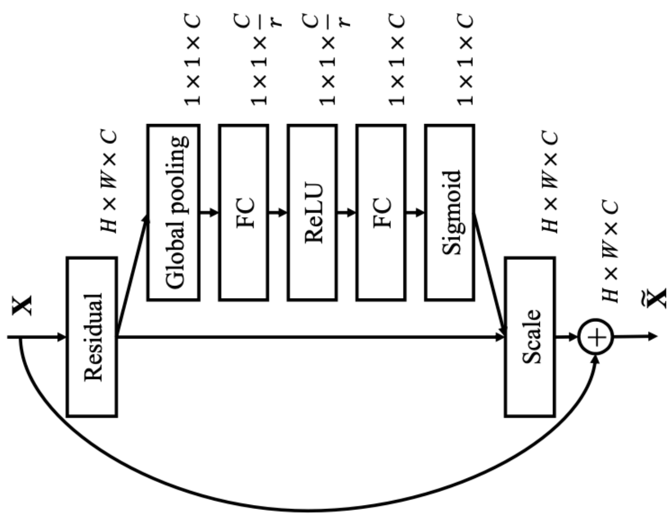
Input Weights Output

Shift

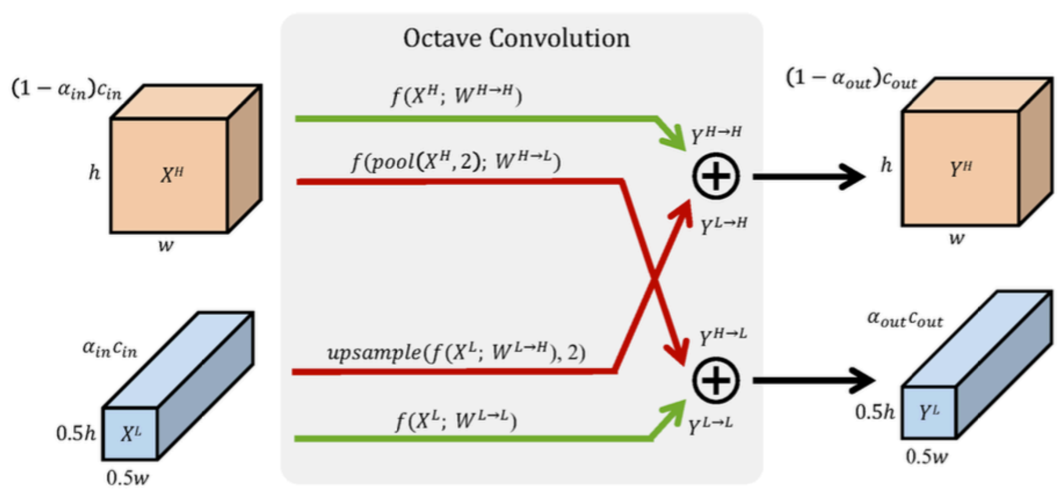


Input Weights Output

Squeeze-Excite

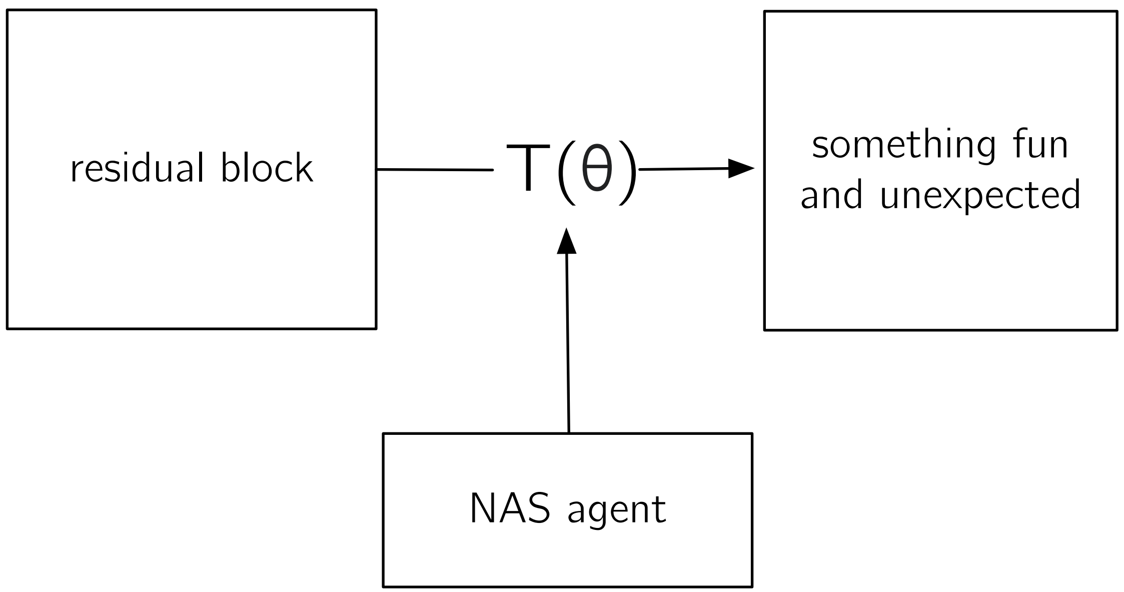


OctaveConv



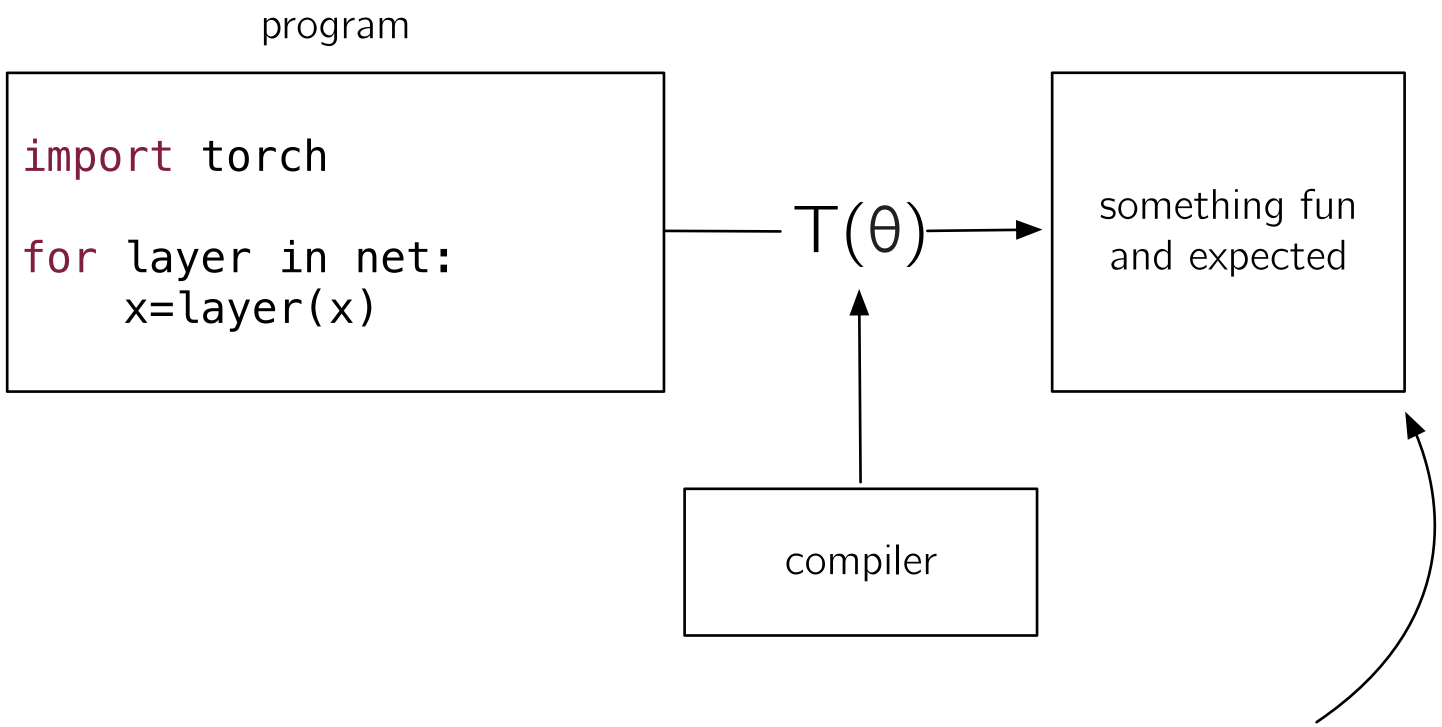
Proposal

Instead of a list of operations, have a list of transformations



Choose a space for T that is more expressive than current NAS, but can still generate forward/backward implementations.

Neural networks as programs

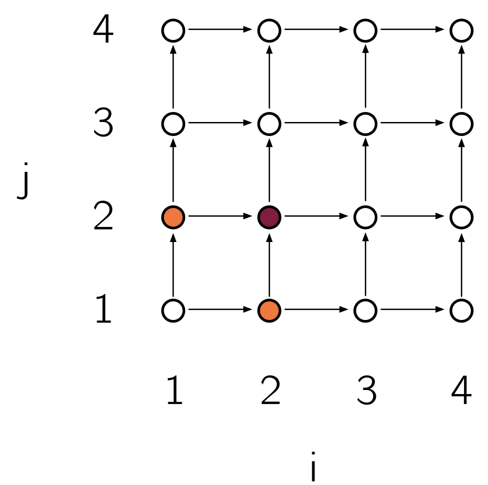


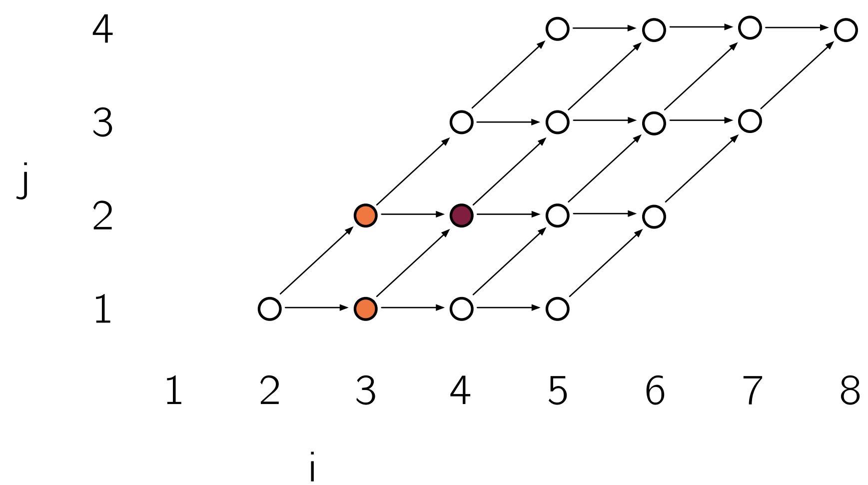
Output program must be semantically equivalent

Program Equivalence as Affine Transformation

```
for i in range(1,N):
  for j in range(1,N):
    A[i,j] = f(A[i-1,j], A[i,j-1])
```

```
for i in range(1,N):
  for j in range(1,N):
    A[i+j,j] = f(A[i+j-1,j], A[i+j,j-1])
```

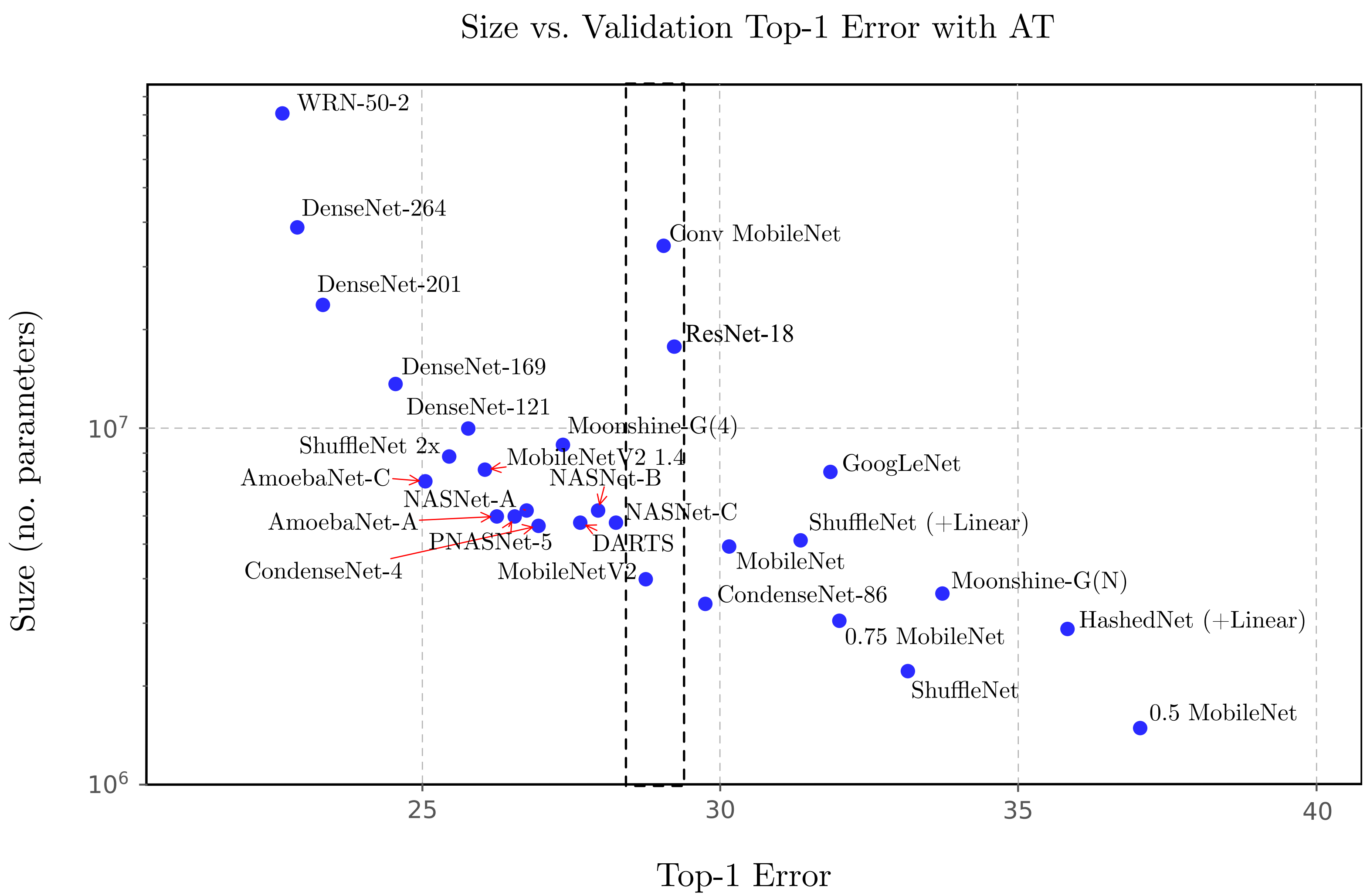




T({i -> i+j, j -> j})

```
.skew(I, +j)
```

Almost program equivalence



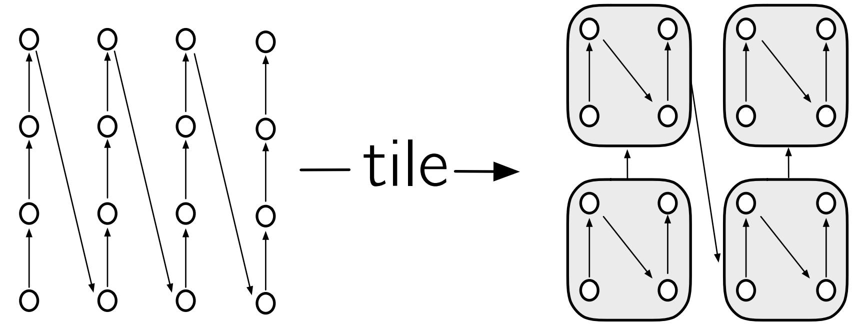
source: Gavin

In neural network land, semantic equivalence really means "just as accurate"

Transformation space

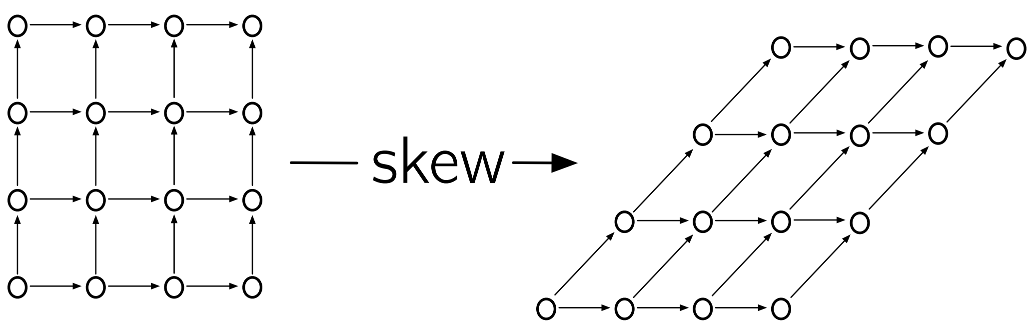

```
for i in range(I):
  for j in range(J):
    A[i][j] += B[i][j]
```

```
for j in range(J):
  for i in range(I):
    A[i][j] += B[i][j]
```



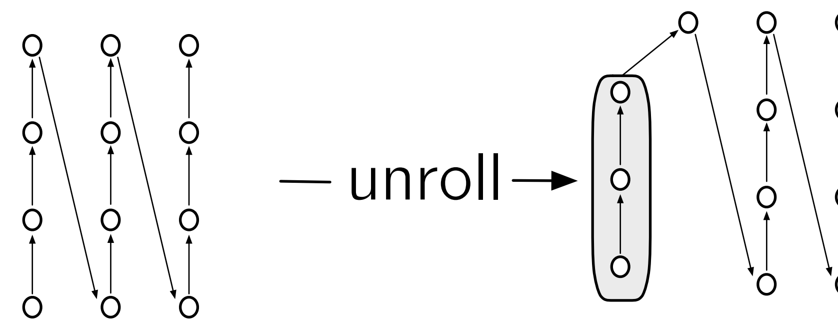

```
for i in range(I):
  for j in range(J):
    A[i][j] += B[i][j]
```

```
@kernel
for i in range(I’):
  for j in range(J’):
    A[i][j] += B[i][j]
```




```
for i in range(1,N):
  for j in range(1,N):
    A[i,j] =
    f(A[i−1,j], A[i,j−1])
```

```
for i in range(1,N):
  for j in range(1,N):
    A[i+j,j] =
    f(A[i+j−1,j], A[i+j,j−1])
```




```
for i in range(I):
  for j in range(J):
    A[i][j] += B[i][j]
```

```
for i in range(I):
  A[i][0] += B[i][0]
  A[i][1] += B[i][1]
  A[i][2] += B[i][2]
  for j in range(3,J):
    A[i][j] += B[i][j]
```

NAS-style Additions

```
for i in range(I):
  for j in range(J):
    A[i,j] += B[i,j]
```

group →

```
for g in range(G):
 for i in range(I//G, I//G+1):
   for j in range(J//G, J//G+1):
     A[i,j] += B[i,j]
```

```
for i in range(I):
  for j in range(J):
    A[i,j] += B[i,j]
```

bottleneck →

```
for i in range(I//B):
   for j in range(J):
     A[i,j] += B[i,j]
```

```
for i in range(I):
  for j in range(J):
    A[i,j] += B[i,j]
```

split →

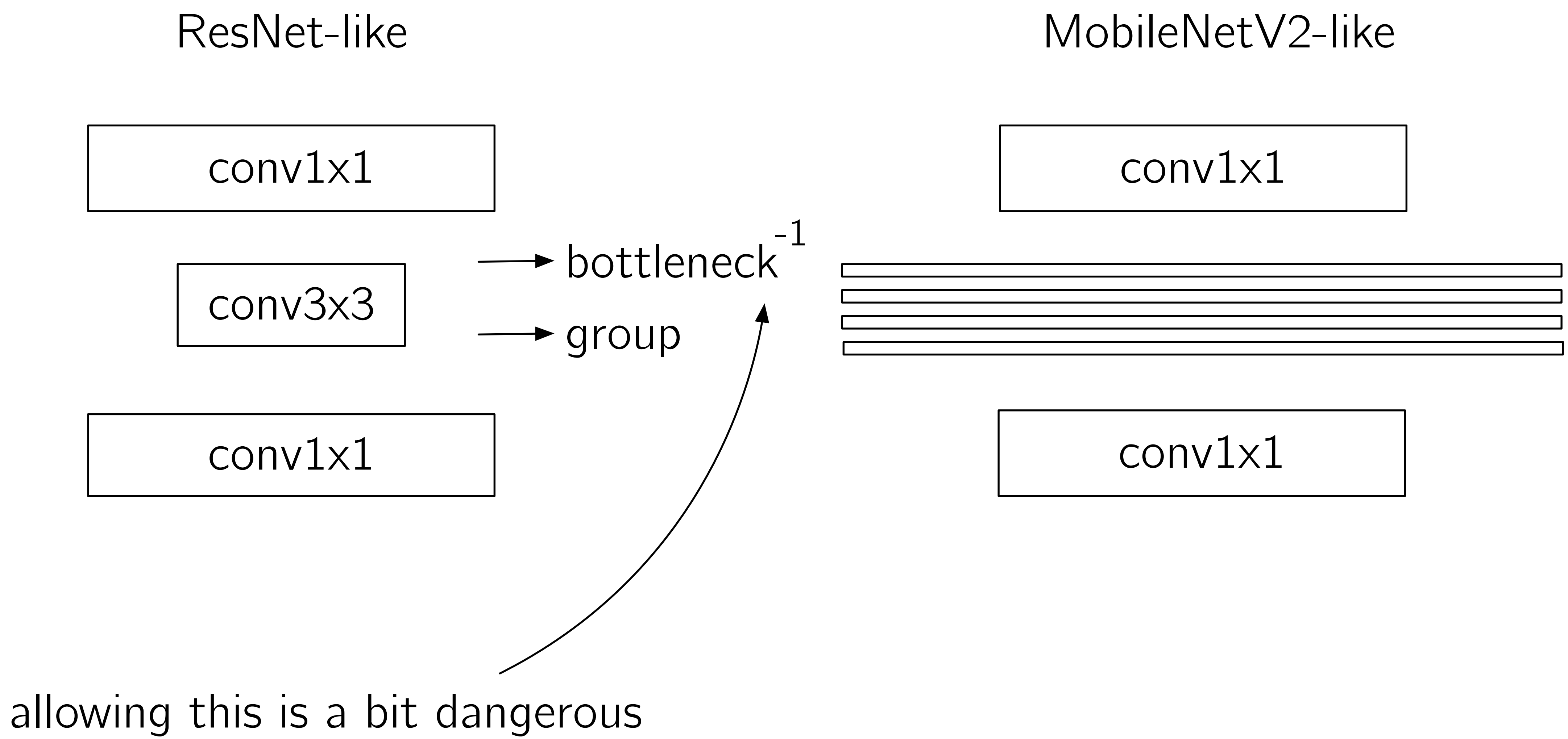
```
for i in range(I//2):
   for j in range(J//2):
     A[i,j] += B[i,j]

for i in range(I//2,I):
   for j in range(J//2,J):
     A[i,j] += B[i,j]
```

Convolution

```
for co in range(CO):
 for ci in range(CI):
  for oh in range(OH):
   for ow in range(OW):
    for kh in range(KH):
     for kw in range(KW):
       O[co][oh][ow] +=
              I[ci][oh+kh][ow+kw] *
              W[co][ci][kh][kw]
```

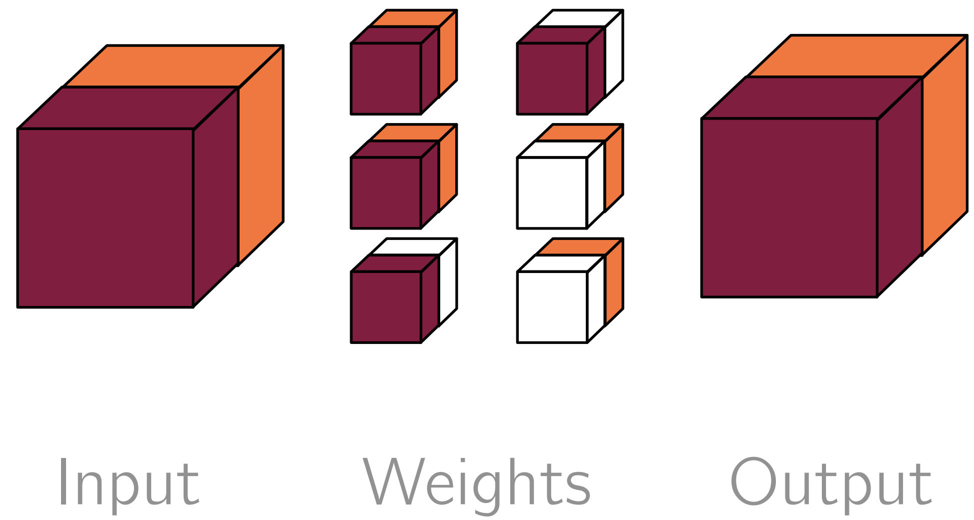
Example 1: ResNet to MobileNetV2



Example 2: Inventing new operations

conv3x3 → unroll

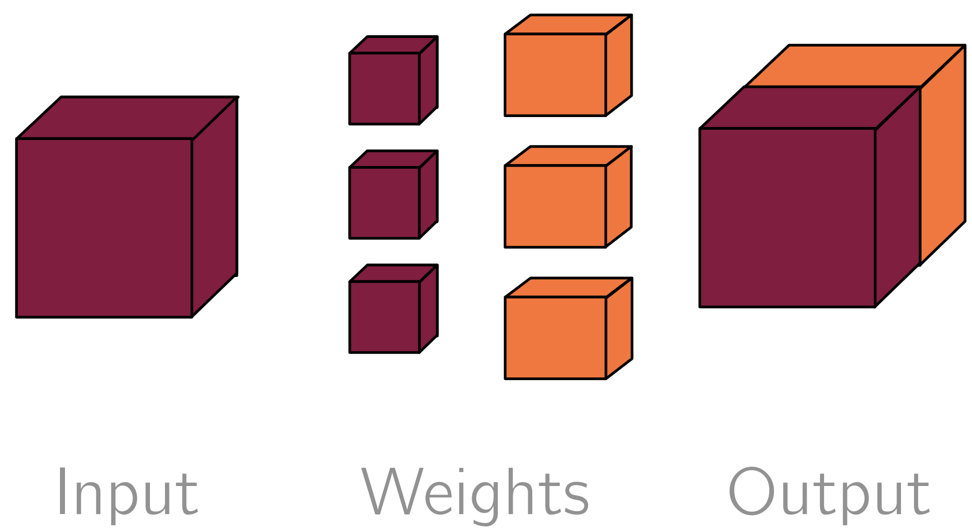
→ group



conv3x3 → interchange

→ split

→ bottleneck^{-1}



Thoughts

What is a minimal set of interesting transformations?

Can we use our polyhedral representations to generate fast forward/backward implementations?

Is there a less ambitious alternative?