FILTRA: Rethinking Steerable CNN by Filter Transform Appendix

A. Verification of Lemma 1 on (23)

(23) can be verified to follow Lemma 1 as:

$$
\begin{aligned}
&\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(\phi+\theta_{i_1}) && (101)\\
&= \mathrm{diag}\left(P(i_1)\mathsf{K}\right)\beta_k, \quad \text{c.f. (18c)} && (102)\\
&= P(i_1)\,\mathrm{diag}(\mathsf{K})P(i_1)^{-1}\beta_k && (103)\\
&= \rho^{\mathrm{C}_N}_{\mathrm{reg}}(g)\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}\psi_{0,k}(g)^{-1}, \quad \text{c.f. (12a).} && (104)
\end{aligned}
$$

We can also verify this for

$$\overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}} = \mathrm{diag}(\overline{\mathsf{K}})\beta_k. \qquad (105)$$

B. Verification of Lemma 1 on (24)

First note it is easy to verify that for $i_0=0$, i.e. $g=(0,i_1)$, the Lemma 1 holds in the same way as (101),

$$\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}(\phi+\theta) = \rho^{\mathrm{D}_N}_{\mathrm{reg}}(g)\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}\psi_{j,k}(g)^{-1}. \qquad (106)$$

We then generalize (21) on $\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}$ and $\overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}}$ given a reflected action $g=(1,i_1)$:

$$
\begin{aligned}
&\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(-\phi+\theta_{i_1}) && (107a)\\
&= \mathrm{diag}\left(\mathsf{K}(-\phi+\theta_{i_1})\right)\beta_k && (107b)\\
&= B(i_1)\,\mathrm{diag}(\overline{\mathsf{K}})B(i_1)^{-1}\beta_k, \quad \text{c.f. (12b)} && (107c)\\
&= B(i_1)\,\mathrm{diag}(\overline{\mathsf{K}})\beta_k\psi_{0,k}(g)^{-1} && (107d)\\
&= -B(i_1)\,\mathrm{diag}(\overline{\mathsf{K}}^{\mathrm{C}_N})\beta_k\psi_{1,k}(g)^{-1} && (107e)\\
&= B(i_1)\,\mathrm{diag}(\mathsf{K}^{\mathrm{C}_N})\beta_k\psi_{0,k}(g)^{-1} && (107f)\\
&= -B(i_1)\,\mathrm{diag}(\mathsf{K}^{\mathrm{C}_N})\beta_k\psi_{1,k}(g)^{-1}. && (107g)
\end{aligned}
$$

Note that (107e) and (107g) both have two equivalent forms denoted with $\psi_{0,k}(g)$ and $\psi_{1,k}(g)$ respectively. Now we can show $\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}$ follows Lemma 1 for $j=0$, $i_0=1$, i.e. $g=(1,i_1)$ as:

$$
\begin{aligned}
&\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}(-\phi+\theta_{i_1}) && (108a)\\
&= \left[\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(-\phi+\theta_{i_1})^\top \quad \overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(-\phi+\theta_{i_1})^\top\right]^\top && (108b)\\
&= \Big[B(i_1)\,\mathrm{diag}(\overline{\mathsf{K}}^{\mathrm{C}_N})\beta_k\psi_{0,k}(g)^{-1}\\
&\qquad B(i_1)\,\mathrm{diag}(\mathsf{K}^{\mathrm{C}_N})\beta_k\psi_{0,k}(g)^{-1}\Big]^\top, && (108c)\\
&\qquad\qquad \text{c.f. (107e)}\\
&= \rho^{\mathrm{D}_N}_{\mathrm{reg}}(g)\left[{\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}}^\top \quad {\overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}}}^\top\right]^\top\psi_{0,k}(g)^{-1}, && (108d)\\
&\qquad\qquad \text{c.f. (23), (105)}\\
&= \rho^{\mathrm{D}_N}_{\mathrm{reg}}(g)\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}\psi_{0,k}(g)^{-1}. && (108e)
\end{aligned}
$$

The verification is similar for $j=1$, $i_0=1$, i.e. $g=(1,i_1)$:

$$
\begin{aligned}
&\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}(-\phi+\theta_{i_1}) && (109a)\\
&= \left[\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(-\phi+\theta_{i_1})^\top \quad -\overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}}(-\phi+\theta_{i_1})^\top\right]^\top && (109b)\\
&= \Big[-B(i_1)\,\mathrm{diag}(\overline{\mathsf{K}}^{\mathrm{C}_N})\beta_k\psi_{1,k}(g)^{-1}\\
&\qquad B(i_1)\,\mathrm{diag}(\mathsf{K}^{\mathrm{C}_N})\beta_k\psi_{1,k}(g)^{-1}\Big]^\top, && (109c)\\
&\qquad\qquad \text{c.f. (107e)}\\
&= \rho^{\mathrm{D}_N}_{\mathrm{reg}}(g)\left[{\mathsf{K}^{\mathrm{C}_N}_{k\to\mathrm{reg}}}^\top \quad -{\overline{\mathsf{K}}^{\mathrm{C}_N}_{k\to\mathrm{reg}}}^\top\right]^\top\psi_{0,k}(g)^{-1}, && (109d)\\
&\qquad\qquad \text{c.f. (23), (105)}\\
&= \rho^{\mathrm{D}_N}_{\mathrm{reg}}(g)\mathsf{K}^{\mathrm{D}_N}_{j,k\to\mathrm{reg}}\psi_{0,k}(g)^{-1}. && (109e)
\end{aligned}
$$

C. Verification of Lemma 1 on (25)

This kernel can be verified as follows for $g = (0, i_1)$:

$$\mathsf{K}^{\mathrm{C}_N}_{\text{reg}\to\text{reg}}(\phi + \theta_{i_1}) \tag{110a}$$

$$= \left[\mathsf{K}^{\mathrm{C}_N}_{0\to\text{reg}}(\phi + \theta_{i_1}) \cdots \mathsf{K}^{\mathrm{C}_N}_{\lfloor\frac{N}{2}\rfloor\to\text{reg}}(\phi + \theta_{i_1})\right] V^{-1} \tag{110b}$$

$$= \left[\rho^{\mathrm{C}_N}_{\text{reg}}(g)\mathsf{K}^{\mathrm{C}_N}_{0\to\text{reg}}\psi_{0,0}(g)^{-1}, \cdots, \rho^{\mathrm{C}_N}_{\text{reg}}(g)\mathsf{K}^{\mathrm{C}_N}_{\lfloor\frac{N}{2}\rfloor\to\text{reg}}\psi_{0,\lfloor\frac{N}{2}\rfloor}(g)^{-1}\right] V^{-1}, \tag{110c}$$

c.f. (104)

$$= \rho^{\mathrm{C}_N}_{\text{reg}}(g)\left[\mathsf{K}^{\mathrm{C}_N}_{0\to\text{reg}} \cdots \mathsf{K}^{\mathrm{C}_N}_{\lfloor\frac{N}{2}\rfloor\to\text{reg}}\right] D^{\mathrm{C}_N} V^{-1} \tag{110d}$$

$$= \rho^{\mathrm{C}_N}_{\text{reg}}(g)\left[\mathsf{K}^{\mathrm{C}_N}_{0\to\text{reg}} \cdots \mathsf{K}^{\mathrm{C}_N}_{\lfloor\frac{N}{2}\rfloor\to\text{reg}}\right] V^{-1} V D^{\mathrm{C}_N} V^{-1} \tag{110e}$$

$$= \rho^{\mathrm{C}_N}_{\text{reg}}(g)\mathsf{K}^{\mathrm{C}_N}_{\text{reg}\to\text{reg}}{\rho^{\mathrm{C}_N}_{\text{reg}}}^{-1}. \tag{110f}$$

D. Minimal Implementation

To illustrate the simplicity of our approach and better explain the tensor operation in the construction of a steerable filter, we list the minimal self-contained PyTorch implementation with only 60 lines of code in this section. Note that this implementation is slightly different from the final version which will be released as open source later.

In the implementation we use `grp` $= (0, N)$ and $(1, N)$ to denote C_N and D_N respectively. `irreps`, `in_irreps` and `out_irreps` denote irreps by a $M \times 2$ matrix, of which each row denotes an irrep (j, k). The steerable convolution operators `IrrepToRegular`, `RegularToIrrep` and `RegularToRegular` preserve the same interface to PyTorch `nn.Conv2d`. The input and output channels are flattened from a structure of `grp`[0] $\times$ `grp`[1] $\times$ `feature_multiplicity`.

```python
import math; import torch as t; import torch.nn.functional as F; LARGE=1e5

def comp_betas(grp, irreps):                                                    # grp: [int, int], irreps: M*2
    angles = t.arange(grp[1]) * math.pi * 2 / grp[1]                            # grp[1]
    angles = t.ger(angles, irreps[:, 1].float())                                # grp[1]*M
    bases = t.stack([angles.cos(), angles.sin()], 1)[None, :, :, :]             # 1*grp[1]*2*M
    flip = ((-1) ** irreps[:, 0]).float()                                       # M
    flip = flip[None, :] ** t.arange(grp[0])[:, None]                           # grp[0]*M
    return flip[:, None, None, :].mul(bases)                                    # grp[0]*grp[1]*2*M

def comp_affine_grid(grp, ker_size):
    angles = t.arange(grp[1]) * math.pi * 2 / grp[1]                            # grp[1]
    c, s = angles.cos(), angles.sin()
    rot = t.stack((c, -s, s, c), 1).view(1, grp[1], 2, 2)                       # 1*grp[1]*2*2
    flip = t.Tensor([[[[1, 0], [0, 1]]], [[[1, 0], [0, -1]]]])[:grp[0]]         # grp[1]*1*2*2
    aff = t.cat((t.matmul(flip, rot), t.zeros(grp + (2, 1))), -1)               # grp[0]*grp[1]*2*3
    grid = F.affine_grid(aff.flatten(0, 1), (grp[0] * grp[1], 1) + ker_size, False)
    disk_mask = (grid.norm(dim=-1) > 1).unsqueeze(-1)                           # (grp[0]*grp[1])*H*W
    return grid.masked_fill(disk_mask, LARGE)

class IrrepToRegular(t.nn.Conv2d):
    def __init__(self, grp, in_irreps, out_mult, ker_size, **kwargs):           # grp: [int, int], in_irreps: M*2
        super(IrrepToRegular, self).__init__(len(in_irreps), out_mult, ker_size, **kwargs)
        self.grp = grp
        self.grid = comp_affine_grid(grp, self.kernel_size)                     # (grp[0]*grp[1])*K*K
        self.expand_coeffs = comp_betas(grp, in_irreps)                         # grp[0]*grp[1]*2*len(in_irreps)

    def expand_filters(self, weight):
        C_o, C_i, H, W = weight.shape
        weight = weight.view(1, -1, H, W).expand(self.grp[0] * self.grp[1], -1, -1, -1)
        steered = F.grid_sample(weight, self.grid, ’bilinear’, ’zeros’, False)  # (grp[0]*grp[1])*(C_o*C_i)*K*K
        steered = steered.view(self.grp + (C_o, 1, -1, H, W))                   # grp[0]*grp[1]*C_o*1*C_i*K*K
        coeffs = self.expand_coeffs.view(self.grp + (1, 2, -1, 1, 1))           # grp[0]*grp[1]*1*2*C_i*1*1
        filters = steered.mul(coeffs)                                           # grp[0]*grp[1]*C_o*2*C_i*K*K
        return filters.flatten(0, 2).flatten(1, 2)                              # (grp[0]*grp[1]*C_o)*(2*C_i)*K*K

    def forward(self, x):
        filters = self.expand_filters(self.weight)
        return F.conv2d(x, filters, self.bias, self.stride,
                self.padding, self.dilation, self.groups)

class RegularToIrrep(IrrepToRegular):
    def __init__(self, grp, in_mult, out_irreps, ker_size, **kwargs):
        super(RegularToIrrep, self).__init__(grp, out_irreps, in_mult, ker_size, **kwargs)

    def expand_filters(self, weight):
        filters = super(RegularToIrrep, self).expand_filters(weight)            # (grp[0]*grp[1]*C_o)*(2*C_i)*K*K
        return filters.permute(1, 0, 2, 3)                                      # (2*C_i)*(grp[0]*grp[1]*C_o)*K*K

class RegularToRegular(IrrepToRegular):
    def __init__(self, grp, in_mult, out_mult, ker_size, **kwargs):
        dct_irreps = t.cartesian_prod(t.arange(grp[0]), t.arange(grp[1] // 2 + 1))
        in_irreps = dct_irreps.repeat_interleave(in_mult, 0)                    # [len(dct_irreps)*in_mult]*2
        super(RegularToRegular, self).__init__(grp, in_irreps, out_mult, ker_size,  **kwargs)
        self.dct_mat = comp_betas(grp, dct_irreps).view(grp[0] * grp[1], -1)[:, :, None]
                                                                                # (grp[0]*grp[1])*(2*len(dct_irreps)
     )*1
    def expand_filters(self, weight):                                           # (grp[0]*grp[1]*out_mult)*
        filters = super(RegularToRegular, self).expand_filters(weight)          #       (2*len(dct_irreps)*in_mult)*
     K*K
        C_o, C_i, H, W = filters.shape                                          # (grp[0]*grp[1]*out_mult)*
        filters = filters.view(C_o, self.dct_mat.shape[1], -1)                  #       (2*len(dct_irreps))*(in_mult
     *K*K)
        filters = F.conv1d(filters, self.dct_mat)                               # (grp[0]*grp[1]*out_mult)*
        return filters.view(C_o, -1, H, W)                                      #       (grp[0]*grp[1]*in_mult)*K*K
```

Listing 1: Minimal implementation of the proposed approach.