Introduction

This is a simple deep learning project focusing on optimizing PyTorch code to perform better with GPU. It is also an exploration with Convolutional Neural Network (CNN) achitecture, specifically ResNet.

Channels Selection Convolution Layer

The learnable weights of a Conv2d module follow the shape:

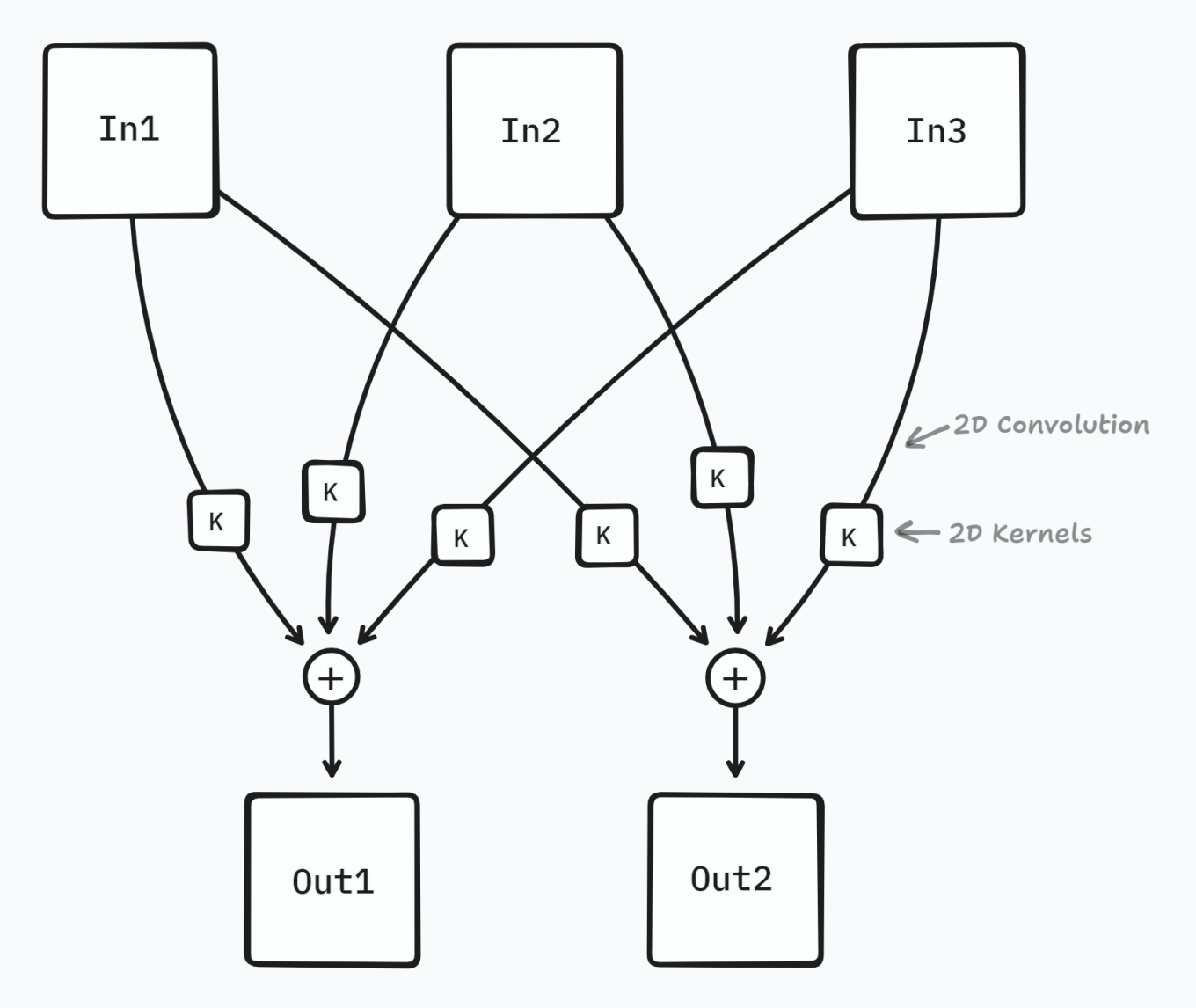
$$ \begin{equation}

```
(\texttt{out\_channels}, \frac{\texttt{in\_channels}}{\texttt{groups}},
\texttt{k\_rows}, \texttt{k\_cols})
```

\end{equation} $$

where, by default, $\texttt{groups}=1$. Pytorch - Conv2d

A typical Conv2d layer (with groups=1) uses individual **sub-kernels** on each input channels then add up the result for an output channel. This repeat for every out-channels in a layer.



Sub-kernels based convolution (torch Conv2d with groups=1)

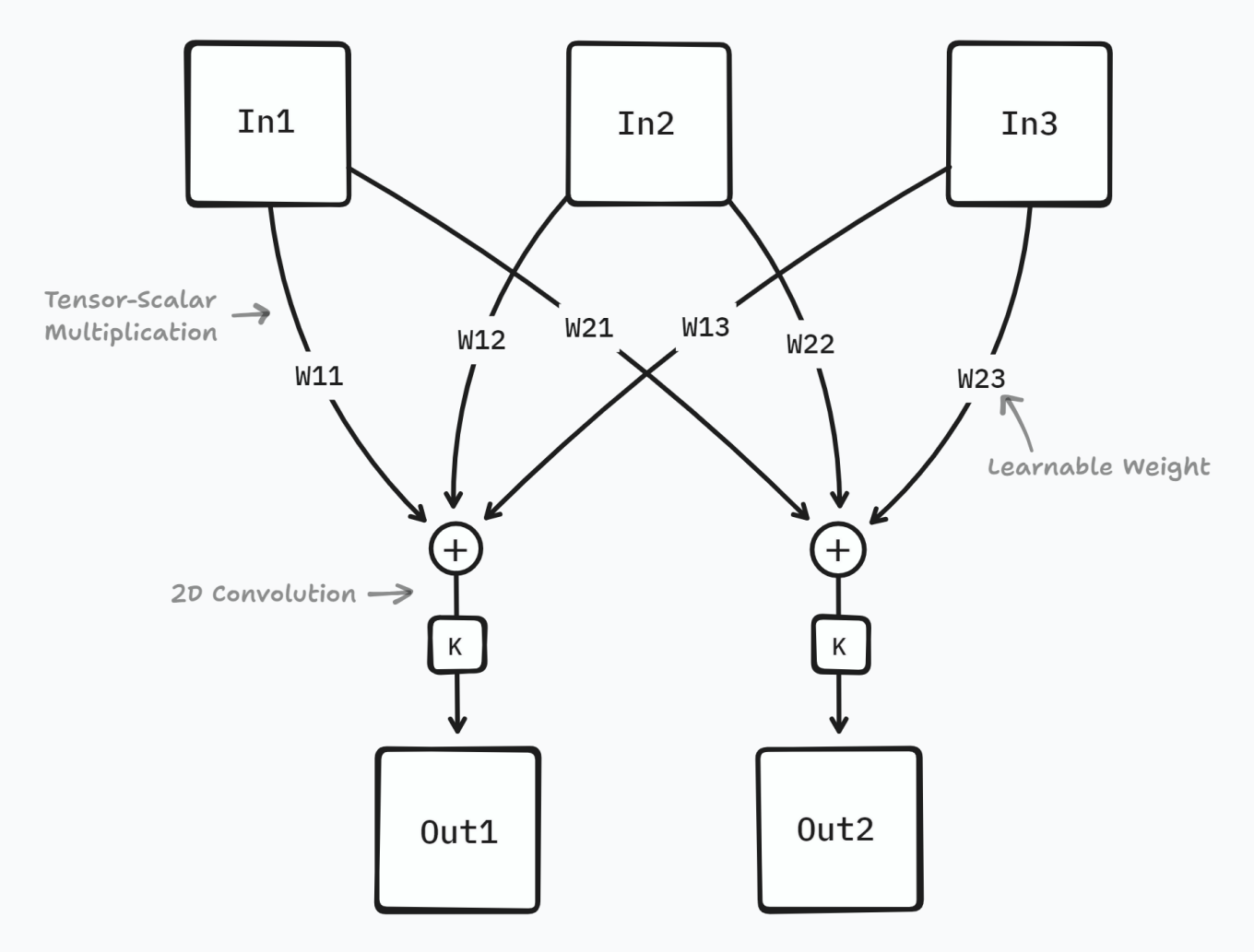
Base on (1) the total number of trainable parameters (not including bias) would be:

$$ \begin{equation}

\texttt{in_channels} \times \texttt{out_channels} \times \texttt{k_rows} \times \texttt{k_cols}

\end{equation} $$

Instead of applying multiple sub-kernels for each output channels, all input channels is weighted sum multiple times for the total of output channels. Each weighted sum uses different set of learnable weights to achieve the idea of input **channels selection** for specific output channels. Individual kernels (with the size of output channels) is then applied to each weighted sum channels.



Channels selection convolution (weighted sum over the input channels for each output channels)

The total learnable parameters for this convolution layer is (without bias):

$$ \begin{equation}

\texttt{in_channels} \times \texttt{out_channels} + \texttt{out_channels} \times K

\end{equation} $$

where $K= \texttt{k_rows} \times \texttt{k_cols}$. Expression (2) is larger than (3) for any $\texttt{in_channels} > \frac{K}{K-1}, K\neq0$. For a typical 3×3 kernels layer, the input channels need to be larger than $9/8 \approx 1$.

Proof: $$ \begin{split}

\texttt{assuming } (1) & > (2) \ \ I\cdot O\cdot K & > I\cdot O + O \cdot K \ I\cdot K & > I + K \ I(K - 1) & > K \ I & > \frac{K}{K - 1}, K \neq 0 \

\end{split} $$

Implementation

```
import torch.nn as nn

class WeightedSumConv2d(nn.Module):
    def __init__(self, in_channels, out_channels):
        # use pytorch Linear layer for channels selection (weighted sum)
        self.fc = nn.Linear(
            in_channels,
            out_channels,
            bias=False)

        # use pytorch Conv2d with groups=out_channels for convolution
        self.conv2d = nn.Conv2d(
            out_channels,
            out_channels,
            # additional arguments
            groups=out_channels)

    # (N = batch size, C_in = in_channels, C_out = out_channels)
    #                                   shape of x:
    def forward(self, x):               # (N, C_in,  W, H    )
        x = x.permute(0, 2, 3, 1)       # (N, W,     H, C_in )
        x = self.fc(x)                  # (N, W,     H, C_out)
        x = x.permute(0, 3, 1, 2)       # (N, C_out, W, H    )
        return conv2d(x)
```

Test Run

The idea of channels selection is tested on the dog-vs-cat-vs-bird dataset [3] against the sub-kernels convolution. The dataset consists of 12k training images divided evenly among three classes (dog, cat, and bird), each input image has the dimension of 32x32.

There was many iterations between model shapes and depth but the final implementation settle with the following configuration:

```
flowchart-elk TD
    classDef empty width:0px;
    classDef box height:35px,line-height:12px,text-align:center;
    A[3x3 conv, 16]:::box
    B01[3x3 conv, 16]:::box
    B02[3x3 conv, 16]:::box
    B0E[ ]:::empty
    B11[3x3 conv, 16]:::box
    B11[3x3 conv, 16]:::box
    B12[3x3 conv, 16]:::box
    B1E[ ]:::empty
    B21[3x3 conv, 32]:::box
    B22[3x3 conv, 32]:::box
    B2E[ ]:::empty
    B31[3x3 conv, 32]:::box
```

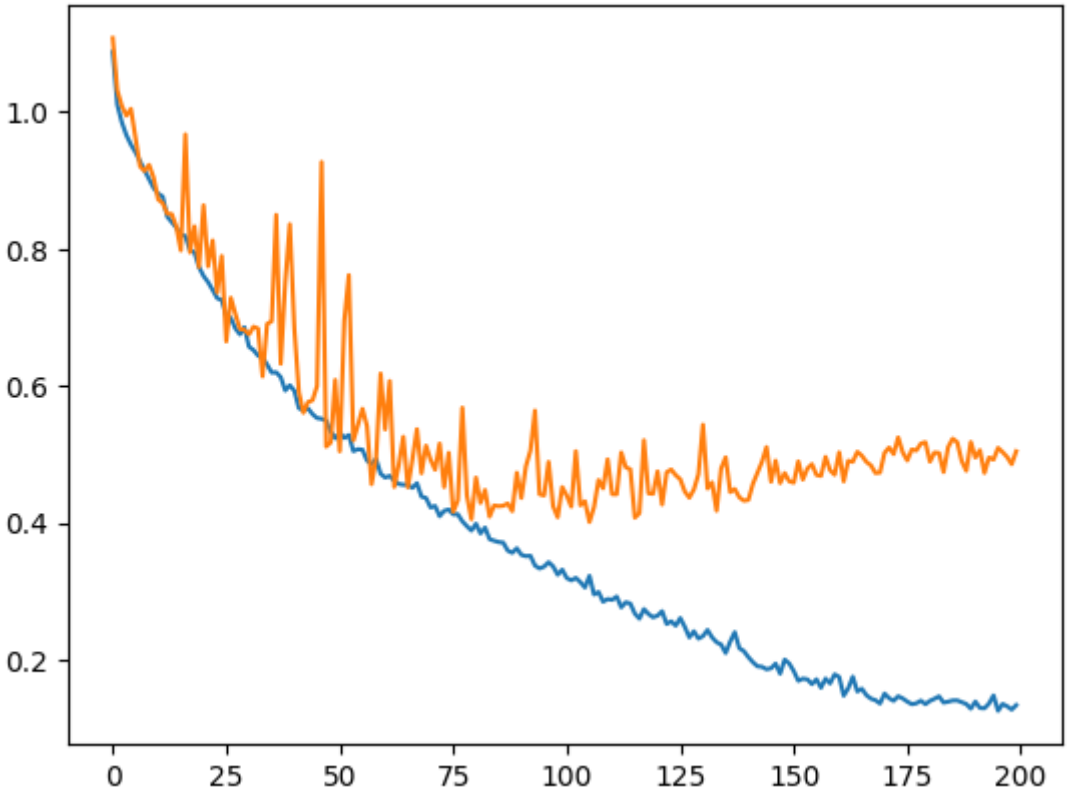
```
    B32[3x3 conv, 32]:::box
    B3E[ ]:::empty
    B41[3x3 conv, 64]:::box
    B42[3x3 conv, 64]:::box
    B4E[ ]:::empty
    B51[3x3 conv, 64]:::box
    B52[3x3 conv, 64]:::box
    B5E[ ]:::empty
    C[AvgeragePool2d]:::box
    D[Dropout 0.2]:::box
    E[FC 64x3]:::box

    A --- B01 --- B02 --- B0E
    A --- B0E
    B0E --- B11 --- B12 --- B1E
    B0E ---> B1E
    B1E --- B21 --- B22 --- B2E
    B1E -.-> B2E
    B2E --- B31 --- B32 --- B3E
    B2E --> B3E
    B3E --- B41 --- B42 --- B4E
    B3E -.-> B4E
    B4E --- B51 --- B52 --- B5E
    B4E --- B5E
    B5E --- C ---> D --- E
```

Each conv are: sub-kernels/channel-selection + batchnorm2d

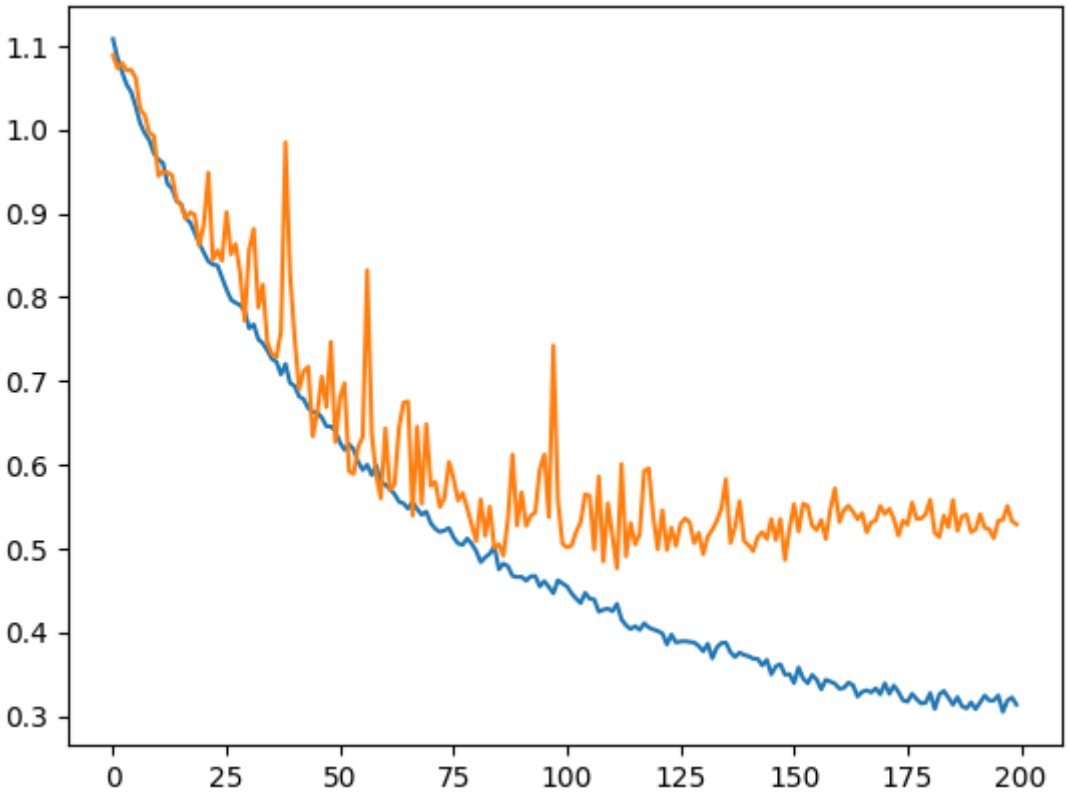
This ResNet inspired model is implemented twice for each type of convolution layer. Jupyter notebooks for both implementations can be found in `notebooks/`.

Models	**Channels selection + Kernels**	**Sub Kernels**
Parameters (custom)	27,363	174,787
Overfitting Epoch	~80 (~50 val loss)	~80 (~0.42 val loss)
Min Test Loss	0.47613 (train: 0.43)	0.40116 (train: 0.32)
Submission Score	81.53%	84.17%
ResNet18 Parameters (performance TBD)	~1,958,824	11,689,512 [2]

> [!NOTE] Learnable parameters for ResNet18 with channels selection is calculated by replacing all Conv2d module with the WeightedSumConv2d module.



sub-kernels convolution based (blue: train losses, orange: validation losses)



channels selection convolution based (blue: train losses, orange: validation losses)

Conclusion

Training speed (pytorch)

Although the Conv2d with channels selection required less computations with lower numebr of learnable parameters, the training time take around twice longer (for small epochs) than with normal Conv2d layer.

Runs	Channels selection time (s/10epochs)	Sub kernels (s/10epochs)
1	85.455	48.41
2	85.3	49.54
3	84.816	47.89
4	84.854	47.66
5	85	47.15
Avg (s/epoch)	8.5085	4.81286

This might be due to the unoptimized and, in a way, unconventional operations required (tensor weighted sum, etc).

> [!NOTE] Using `torch.compile` might help improve the the training speed but Kaggle's GPU was older than the required version for `torch.compile`. Thus it is worth running the model on a different machine.

The ability to learn

It's natural to think that channels selection should performs worse than ordinary sub-kernels-based conv2d, which it did during the dog-cat-bird test. But a bigger problem occur during the experiments. The model with channels selection conv2d seem to be incapable of overfitting and unable to surpass a certain loss value, . This might depend on the high-level achitecture of the model and many other imperfection.

Base on the performance from the dog-cat-bird dataset [3], it is self to say that the channel selection conv2d model perform worst than the ordinary sub-kernels convolution based model. The idea of channels selection convolution need to be tested with different benchmark/dataset.

It is also worth noting that throughout the experiment, different model shapes and hyperparameter were tested. But most, if not all, of the run has shown that the channels selection convolution is incapable to surpass a certain validation loss. Even in the case of intentionally overfitting the dataset. Further understanding of CNN required to tune a model with this convolution layers.