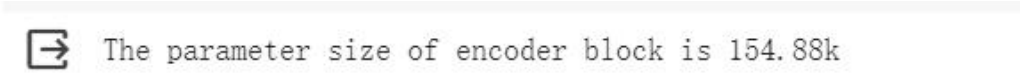
109511085 洪鈺翔 lab03

TASK1 result:

```
Train:   1% 13/2000 [00:00<00:36, 53.86 step/s, accuracy=0.78, loss=1.20, step=5e+4]Step 50000, best model saved. (accuracy=0.6299)
Train: 100% 2000/2000 [02:42<00:00, 12.31 step/s, accuracy=0.66, loss=1.50, step=52000]
Valid: 100% 5664/5667 [00:16<00:00, 346.46 uttr/s, accuracy=0.64, loss=1.62]
Train: 100% 2000/2000 [02:44<00:00, 12.13 step/s, accuracy=0.50, loss=2.35, step=54000]
Valid: 100% 5664/5667 [00:16<00:00, 344.26 uttr/s, accuracy=0.64, loss=1.64]
Train: 100% 2000/2000 [02:47<00:00, 11.93 step/s, accuracy=0.66, loss=1.75, step=56000]
Valid: 100% 5664/5667 [00:16<00:00, 341.92 uttr/s, accuracy=0.63, loss=1.65]
Train: 100% 2000/2000 [02:42<00:00, 12.33 step/s, accuracy=0.72, loss=1.40, step=58000]
Valid: 100% 5664/5667 [00:16<00:00, 344.22 uttr/s, accuracy=0.66, loss=1.58]
```

TASK1 parameter:



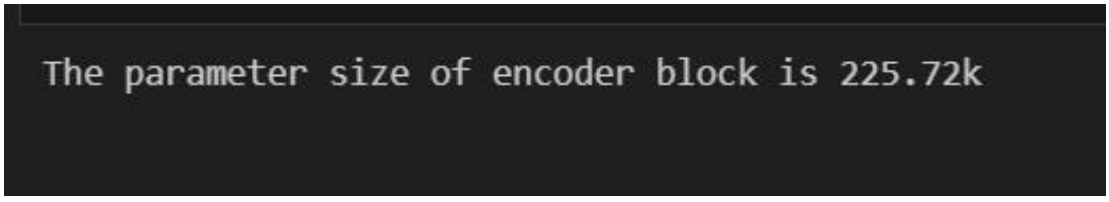
TASK1 layer number:

```
self.encoder_layer = TransformerEncoderLayer(d_model, dim_feedforward=10,nhead=1) #your own transformer encoder layer
self.encoder = TransformerEncoder(d_model, self.encoder_layer,num_layers=3)#your own transformer encoder
# Project the the dimension of features from d_model into speaker nums.
self.pred_layer = nn.Sequential(
    nn.Linear(d_model, d_model),
    nn.ReLU(),
    nn.Linear(d_model, n_spks),
)
```

TASK2 result:

```
Train:   0% 0/2000 [00:00<?, ? step/s]
Train:   0% 2/2000 [00:00<06:45,  4.93 step/s, accuracy=0.81, loss=0.99, step=3e+4]Step 30000, best model saved. (accuracy=0.6873)
Train: 100% 2000/2000 [03:00<00:00, 11.09 step/s, accuracy=0.78, loss=0.92, step=32000]
Valid: 100% 5664/5667 [00:17<00:00, 327.24 uttr/s, accuracy=0.70, loss=1.34]
Train: 100% 2000/2000 [02:57<00:00, 11.26 step/s, accuracy=0.75, loss=0.70, step=34000]
Valid: 100% 5664/5667 [00:17<00:00, 320.75 uttr/s, accuracy=0.70, loss=1.30]
```

TASK2 parameter:



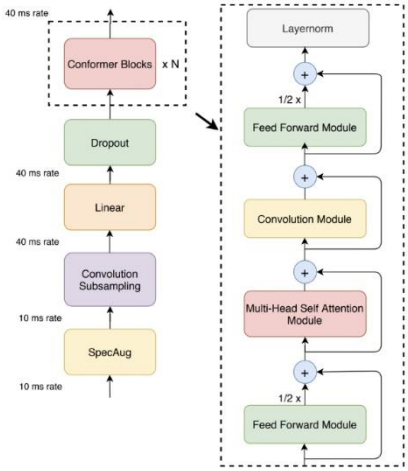
TASK2 layer number:

```
self.encoder_layer = TransformerEncoderLayer(d_model, dim_feedforward = 500,nhead = 1) #your own transformer encoder layer
self.encoder = TransformerEncoder(d_model, self.encoder_layer,num_layers = 3)#your own transformer encoder
# Project the the dimension of features from d_model into speaker nums.
self.pred_layer = nn.Sequential(
    #nn.Linear(d_model, d_model),
    #nn.ReLU(),
    nn.BatchNorm1d(d_model),
    nn.Linear(d_model, n_spks),
)
```

TASK2 model:

我選擇 conformer 作為任務二的模型，conformer 與 transformer 的不同，在於 conformer 加入 convolution 在 encoder layer 中，attention 可以得到全局的關聯性，convolution 則幫助我們得到一部分的關聯性，一個 globle 一個 local 理論可以幫助我們得到更多資訊。

因此 conformer 在 feedforward 之前或之後，加入捲積層，等於直接對 attention 的輸出做 local 訊息的強化。



而我選擇 conformer 的原因，除了助教推薦外，因為 branchformer 本身是 conformer 的延伸，他同樣要獲取 global 跟 local 的訊息，但是他將原本的串接方式，改為平行化，理論上可以讓模型更有彈性，收斂也會更快，但是畢竟是 conformer 的下一代，因此我想先完成 conformer，如果 conformer 無法達到助教的要求，在座 branchformer。

Parameter adjustment:

此次參數調整主要在參數量的調整，可以調整 dim_feedforward 來調整全連接層的參數量，但是我跟同學們都有發現，參數量向上調的過程中，模型的表現並沒有變好，這個問題在 transformer 的時候不明顯，因為 transformer 我們沒有遇到不能收斂的情況，但是到 conformer，這就是嚴重的問題因為真的有收斂不起來準確度下降到零的情況。

後來，我們發現，需要將 classifier 的線性層作調整，需要再前面再連接正規化層，如此一來大量的參數才有意義，我們的推測是因為參數量大，所以有可能讓數值爆掉，因為 relu 也沒有上限，所以可能太大的值影響力過大，導致其他訊息對結果沒有影響，才讓結果沒差或無法收斂。

CODE:

```python
class TransformerEncoderLayer(nn.Module):
    def __init__(self, d_model, dim_feedforward, nhead, dropout = 0.1):
        super().__init__()
        self.enc_self_attn = MultiHeadAttention(d_model , nhead)
        self.Norm1 = nn.LayerNorm(d_model)
        self.Norm2 = nn.LayerNorm(d_model)
        self.feedforward = nn.Sequential(
            nn.Linear(d_model, dim_feedforward * d_model),
            nn.ReLU(),
            nn.Linear(dim_feedforward * d_model, d_model)
        )
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        attention = self.enc_self_attn(x,x,x)
        x = self.dropout(self.Norm1(attention + x))
        forward = self.feedforward(x)
        out = self.dropout(forward)
        return out
```

```python
class TransformerEncoder(nn.Module):
    def __init__(self,d_model, encode_layer, num_layers):
        super().__init__()
        self.layers = nn.ModuleList(
            [
                encode_layer
                for _ in range(num_layers)
            ]
        )

    def forward(self, x):
        N = x.shape
        for layer in self.layers:
            x = layer(x)
        return x
```

```python
class TransformerEncoderLayer(nn.Module):
    def __init__(self, d_model, dim_feedforward, nhead, dropout = 0.1):
        super().__init__()
        self.enc_self_attn = MultiHeadAttention(d_model , nhead)
        self.Norm1 = nn.LayerNorm(d_model)
        self.Norm2 = nn.LayerNorm(d_model)
        self.feedforward = nn.Sequential(
            nn.Linear(d_model, dim_feedforward),
            nn.ReLU(),
            nn.Linear(dim_feedforward, d_model),
            nn.Dropout(dropout)
        )
        self.conv1d = nn.Conv1d(d_model,2 * d_model, kernel_size = 3, padding = 1, bias = False)
        self.glu = nn.GLU(dim = 1)
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.linear2 = nn.Linear(dim_feedforward, d_model)
        self.Norm3 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)
    def forward(self, x):
        attention = self.enc_self_attn(x,x,x)
        x = self.Norm1(attention + x)
        forward = self.feedforward(x)
        x = x + forward
        x = self.Norm2(x)

        #conv
        residual = x
        x = x.transpose(1,2)
        x = self.conv1d(x)
        x = self.glu(x)
        x = x.transpose(1,2)
        x = self.linear2(F.relu(self.linear1(x)))
        x = self.Norm3(x + residual)
        x = self.dropout(x)
        return x
```

```python
class TransformerEncoder(nn.Module):
    def __init__(self,d_model, encode_layer, num_layers):
        super().__init__()
        self.layers = nn.ModuleList(
            [
                encode_layer
                for _ in range(num_layers)
            ]
        )

    def forward(self, x):
        N = x.shape
        for layer in self.layers:
            x = layer(x)
        return x
```