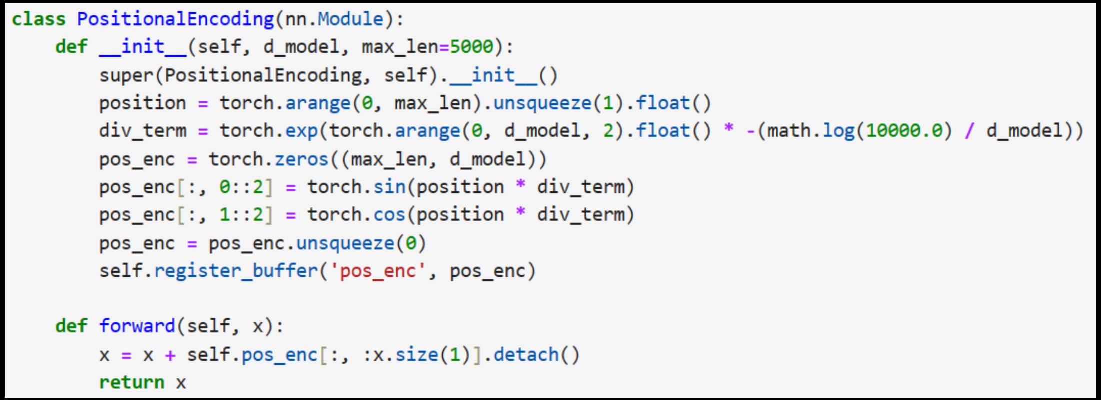
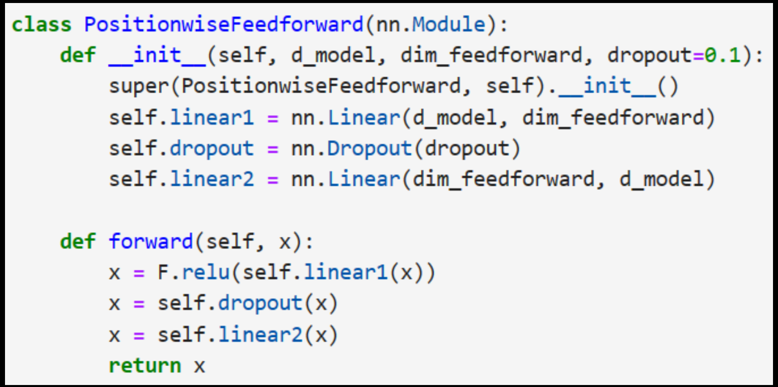
Task 1

○ Vanilla Transformer

Vanilla Transformer的核心是自注意力機制，允許模型在處理序列時動態地關注不同位置的資訊，而不受固定視窗的限制，使Transformer能夠高效地進行平行計算，並且加速了訓練過程。Vanilla Transformer還引入了位置編碼，以處理輸入序列中單詞的位置資訊，解決了Transformer無法處理序列順序的問題。自注意力機制之後通常都會接一個全連接前饋網絡，這樣可以提高模型的非線性能力，還能使模型進行高級的特徵提取，讓模型更好地理解輸入數據中的抽象特徵，從而提高模型的表示能力。



```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        position = torch.arange(0, max_len).unsqueeze(1).float()
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * -(math.log(10000.0) / d_model))
        pos_enc = torch.zeros((max_len, d_model))
        pos_enc[:, 0::2] = torch.sin(position * div_term)
        pos_enc[:, 1::2] = torch.cos(position * div_term)
        pos_enc = pos_enc.unsqueeze(0)
        self.register_buffer('pos_enc', pos_enc)

    def forward(self, x):
        x = x + self.pos_enc[:, :x.size(1)].detach()
        return x
```

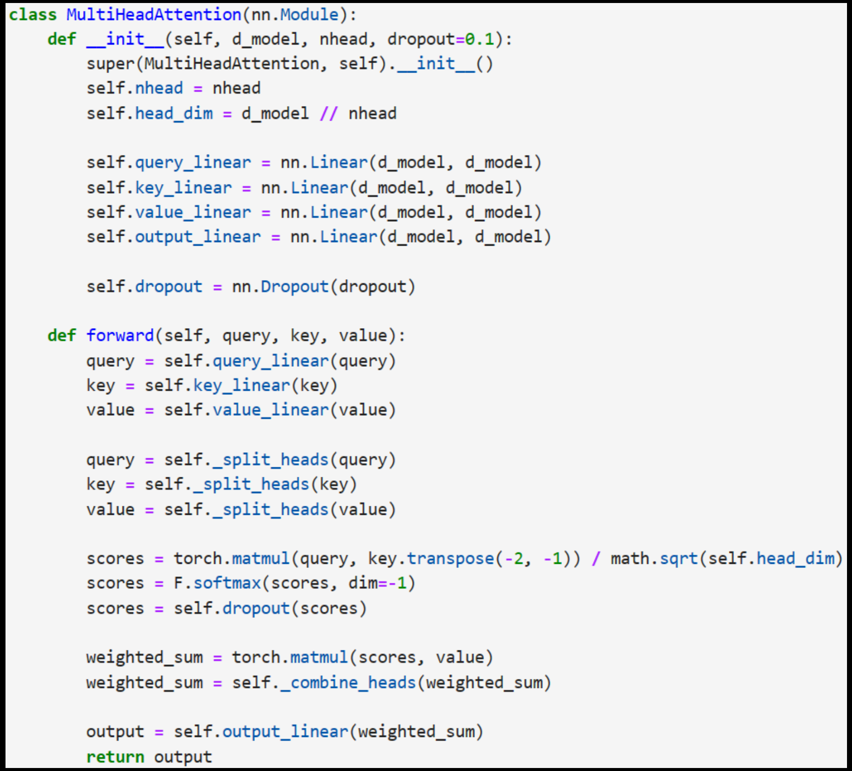


```python
class PositionwiseFeedforward(nn.Module):
    def __init__(self, d_model, dim_feedforward, dropout=0.1):
        super(PositionwiseFeedforward, self).__init__()
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(dim_feedforward, d_model)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = self.dropout(x)
        x = self.linear2(x)
        return x
```

Task 1

- 多頭注意力 (Multi-Head Attention)

多頭注意力也是Vanilla Transformer的其中一個特點，它允許模型同時學習多個不同的關注方向，從而更好地捕捉序列中的不同關係，提高模型的表示能力。除此之外，多頭還可以學習不同的表示，使得模型能夠更全面地理解輸入序列，這有助於處理輸入資料的多樣性。通過允許模型同時關注不同部分的資訊，多頭注意力能夠提高模型對各種輸入模式的泛化性能。

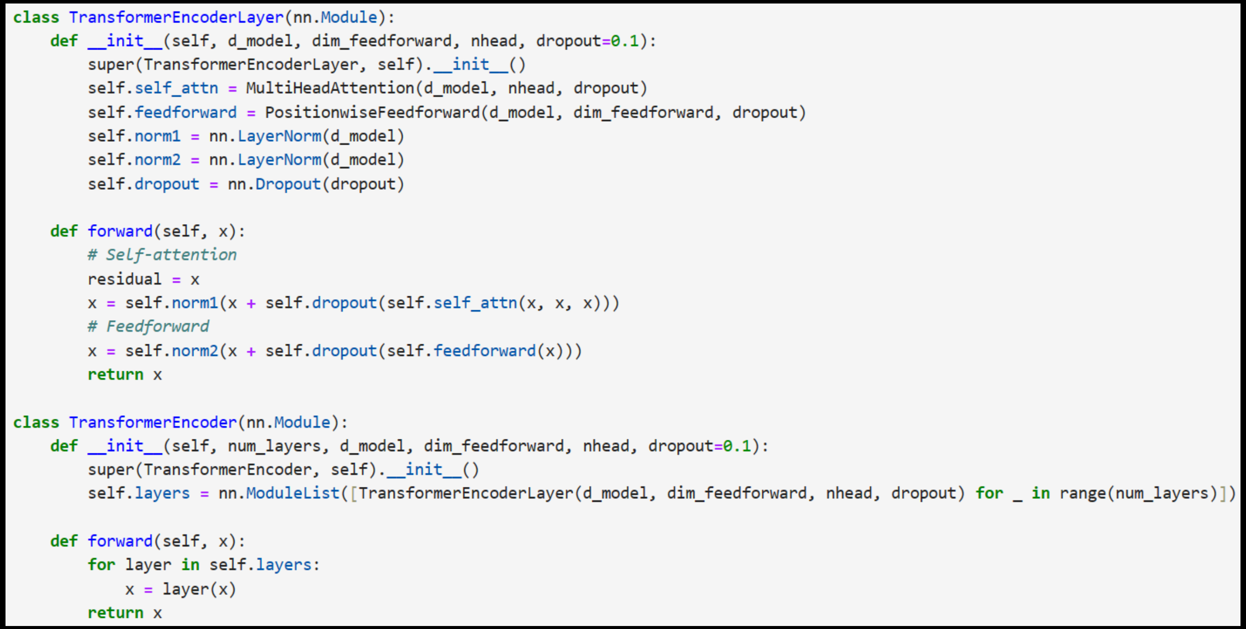


```python
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, nhead, dropout=0.1):
        super(MultiHeadAttention, self).__init__()
        self.nhead = nhead
        self.head_dim = d_model // nhead

        self.query_linear = nn.Linear(d_model, d_model)
        self.key_linear = nn.Linear(d_model, d_model)
        self.value_linear = nn.Linear(d_model, d_model)
        self.output_linear = nn.Linear(d_model, d_model)

        self.dropout = nn.Dropout(dropout)

    def forward(self, query, key, value):
        query = self.query_linear(query)
        key = self.key_linear(key)
        value = self.value_linear(value)

        query = self._split_heads(query)
        key = self._split_heads(key)
        value = self._split_heads(value)

        scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(self.head_dim)
        scores = F.softmax(scores, dim=-1)
        scores = self.dropout(scores)

        weighted_sum = torch.matmul(scores, value)
        weighted_sum = self._combine_heads(weighted_sum)

        output = self.output_linear(weighted_sum)
        return output
```

Task 1

○ 編碼器-解碼器 (Encoder-Decoder)

編碼器-解碼器是Vanilla Transformer重要的一部分，編碼器負責將輸入序列轉換為固定維度的隱藏表示，而解碼器則使用這個表示生成目標序列，這有助於更好地理解序列中的相關資訊。由於編碼和解碼階段的存在，Encoder-Decoder結構能夠有效地處理可變長度的輸入和輸出序列。

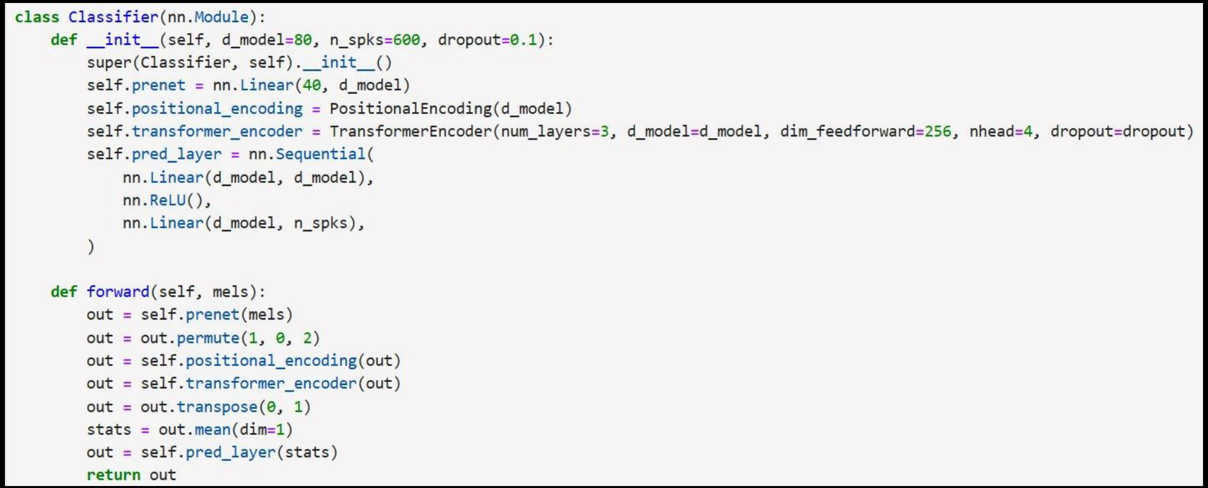


```
class TransformerEncoderLayer(nn.Module):
    def __init__(self, d_model, dim_feedforward, nhead, dropout=0.1):
        super(TransformerEncoderLayer, self).__init__()
        self.self_attn = MultiHeadAttention(d_model, nhead, dropout)
        self.feedforward = PositionwiseFeedforward(d_model, dim_feedforward, dropout)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        # Self-attention
        residual = x
        x = self.norm1(x + self.dropout(self.self_attn(x, x, x)))
        # Feedforward
        x = self.norm2(x + self.dropout(self.feedforward(x)))
        return x

class TransformerEncoder(nn.Module):
    def __init__(self, num_layers, d_model, dim_feedforward, nhead, dropout=0.1):
        super(TransformerEncoder, self).__init__()
        self.layers = nn.ModuleList([TransformerEncoderLayer(d_model, dim_feedforward, nhead, dropout) for _ in range(num_layers)])

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

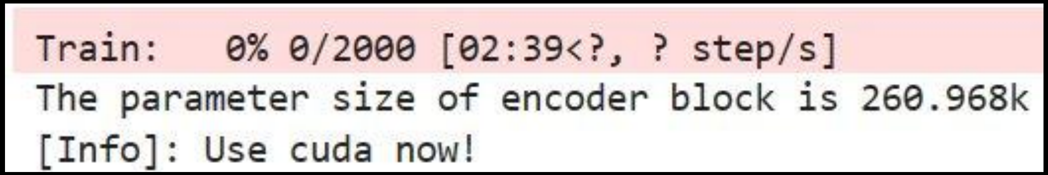
Task 1

○ 參數設置(Parameter Setting)

我的Vanilla Transformer一開始設置的參數都和助教差不多，模型的特徵維度設置為80，代表每個位置的向量將包含80個特徵。然後Transformer編碼器中的層數設置為3（助教規定小於4）。這邊我使用4個頭來計算多頭注意力。至於前饋神經網路的隱藏層維度，我設置為256，表示前饋神經網路中間層的大小為256。最後encoder block的參數大小為260.968k，小於500k，符合助教的條件限制。最終的模型精度達68.01%(>65%)。

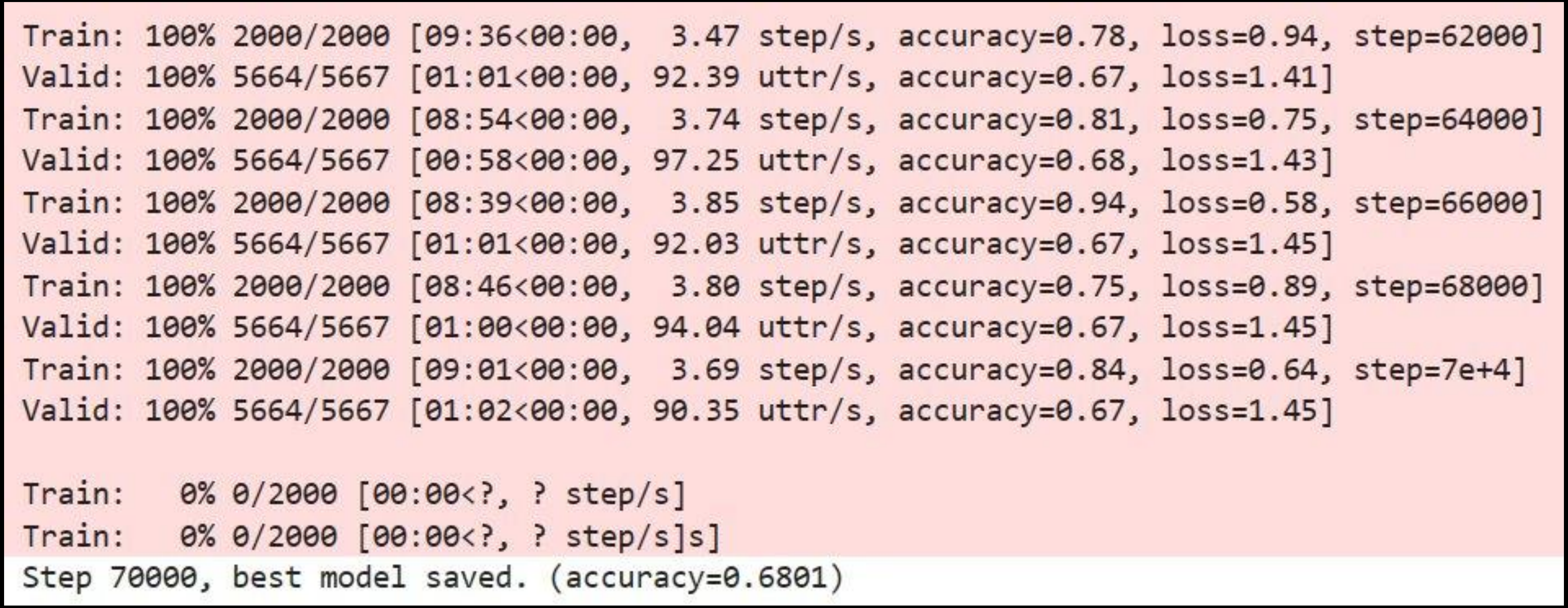


```python
class Classifier(nn.Module):
    def __init__(self, d_model=80, n_spks=600, dropout=0.1):
        super(Classifier, self).__init__()
        self.prenet = nn.Linear(40, d_model)
        self.positional_encoding = PositionalEncoding(d_model)
        self.transformer_encoder = TransformerEncoder(num_layers=3, d_model=d_model, dim_feedforward=256, nhead=4, dropout=dropout)
        self.pred_layer = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.ReLU(),
            nn.Linear(d_model, n_spks),
        )

    def forward(self, mels):
        out = self.prenet(mels)
        out = out.permute(1, 0, 2)
        out = self.positional_encoding(out)
        out = self.transformer_encoder(out)
        out = out.transpose(0, 1)
        stats = out.mean(dim=1)
        out = self.pred_layer(stats)
        return out
```



```
Train:   0% 0/2000 [02:39<?, ? step/s]
The parameter size of encoder block is 260.968k
[Info]: Use cuda now!
```

Task 1 Result

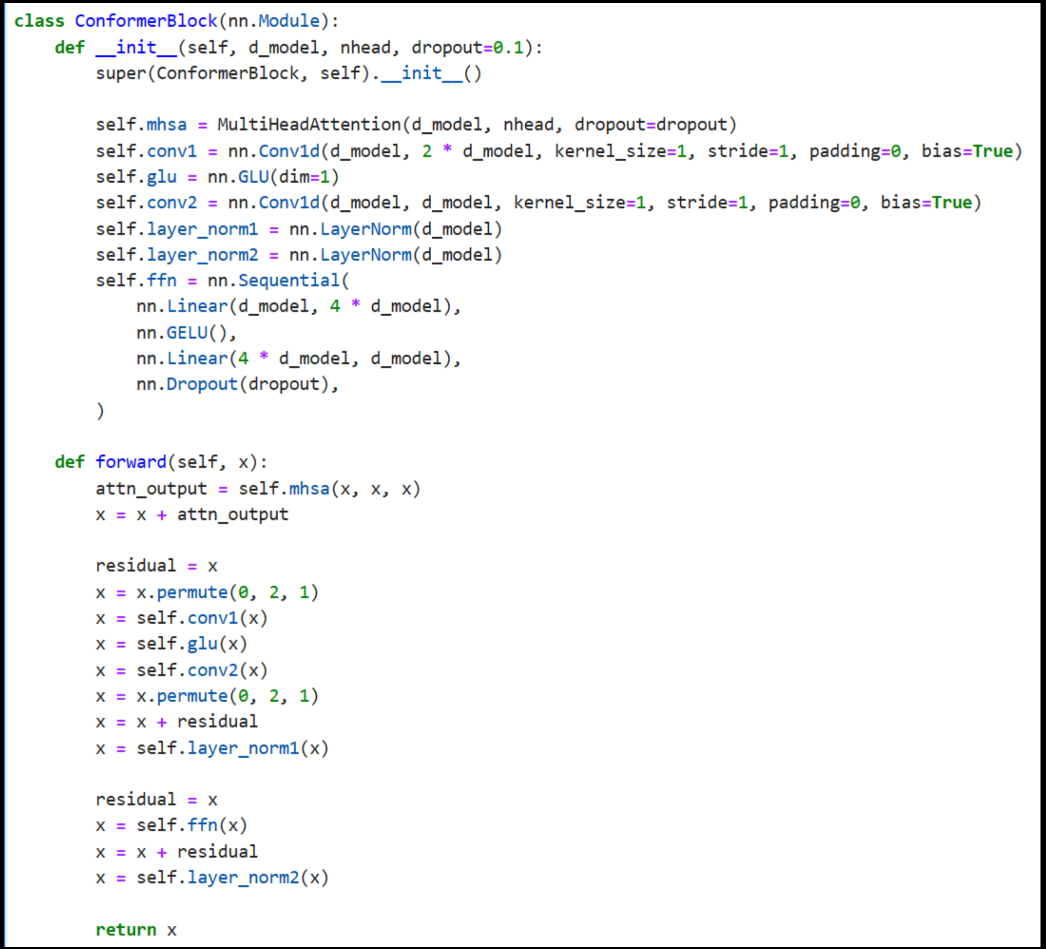


```
Train: 100% 2000/2000 [09:36<00:00,  3.47 step/s, accuracy=0.78, loss=0.94, step=62000]
Valid: 100% 5664/5667 [01:01<00:00, 92.39 uttr/s, accuracy=0.67, loss=1.41]
Train: 100% 2000/2000 [08:54<00:00,  3.74 step/s, accuracy=0.81, loss=0.75, step=64000]
Valid: 100% 5664/5667 [00:58<00:00, 97.25 uttr/s, accuracy=0.68, loss=1.43]
Train: 100% 2000/2000 [08:39<00:00,  3.85 step/s, accuracy=0.94, loss=0.58, step=66000]
Valid: 100% 5664/5667 [01:01<00:00, 92.03 uttr/s, accuracy=0.67, loss=1.45]
Train: 100% 2000/2000 [08:46<00:00,  3.80 step/s, accuracy=0.75, loss=0.89, step=68000]
Valid: 100% 5664/5667 [01:00<00:00, 94.04 uttr/s, accuracy=0.67, loss=1.45]
Train: 100% 2000/2000 [09:01<00:00,  3.69 step/s, accuracy=0.84, loss=0.64, step=7e+4]
Valid: 100% 5664/5667 [01:02<00:00, 90.35 uttr/s, accuracy=0.67, loss=1.45]

Train:   0% 0/2000 [00:00<?, ? step/s]
Train:   0% 0/2000 [00:00<?, ? step/s]s]
Step 70000, best model saved. (accuracy=0.6801)
```

Task 2

- **Conformer**

由於助教課堂上解説Lab 3的時候，強烈推薦我們使用Conformer，因此才選擇該模型來做語音辨識的工作。Conformer是針對Transformer改進的模型，引入了一維卷積層，用於更好地捕捉序列中的局部特徵。這有助於提高模型對輸入序列的細節感知能力。他還替代了Vanilla Transformer中的固定長度的注意力機制，使得模型可以動態地選擇關注輸入序列的哪些部分，提高了模型的效率。這些改進使得Conformer在處理長序列方面更為有效，適合用來做語音辨識的任務。

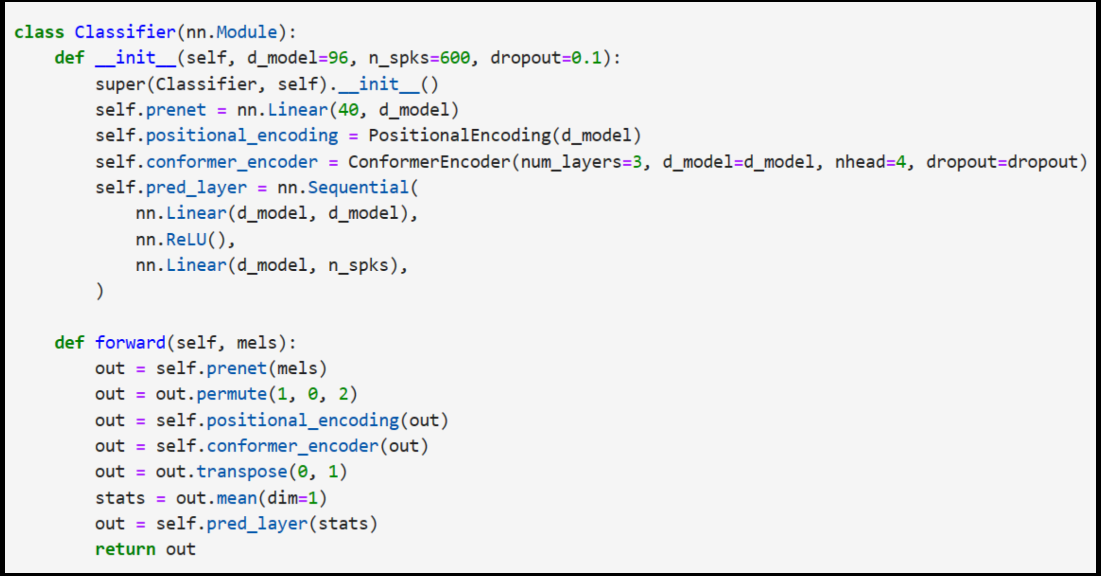


```python
class ConformerBlock(nn.Module):
    def __init__(self, d_model, nhead, dropout=0.1):
        super(ConformerBlock, self).__init__()

        self.mhsa = MultiHeadAttention(d_model, nhead, dropout=dropout)
        self.conv1 = nn.Conv1d(d_model, 2 * d_model, kernel_size=1, stride=1, padding=0, bias=True)
        self.glu = nn.GLU(dim=1)
        self.conv2 = nn.Conv1d(d_model, d_model, kernel_size=1, stride=1, padding=0, bias=True)
        self.layer_norm1 = nn.LayerNorm(d_model)
        self.layer_norm2 = nn.LayerNorm(d_model)
        self.ffn = nn.Sequential(
            nn.Linear(d_model, 4 * d_model),
            nn.GELU(),
            nn.Linear(4 * d_model, d_model),
            nn.Dropout(dropout),
        )

    def forward(self, x):
        attn_output = self.mhsa(x, x, x)
        x = x + attn_output

        residual = x
        x = x.permute(0, 2, 1)
        x = self.conv1(x)
        x = self.glu(x)
        x = self.conv2(x)
        x = x.permute(0, 2, 1)
        x = x + residual
        x = self.layer_norm1(x)

        residual = x
        x = self.ffn(x)
        x = x + residual
        x = self.layer_norm2(x)

        return x
```

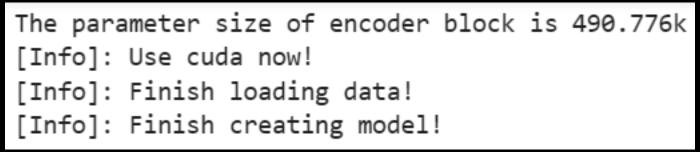
Task 2

- 參數設置 (Parameter Setting)

我的Conformer設置的參數和Task 1的Vanilla Transformer相似，只是將模型的特徵維度改成為96，代表模型中的隱藏層將包含96個特徵。然後Conformer編碼器中的層數依然設置為3（助教規定小於4）。最後encoder block的參數大小為490.776k，小於500k，符合助教的條件限制。最終的模型精度達75.05%(>68%)。我認爲只要在稍微調整一下參數和α(學習率),它應該很容易就能突破80%甚至達到90%，這説明了Conformer真的很適合用來做語音辨識。

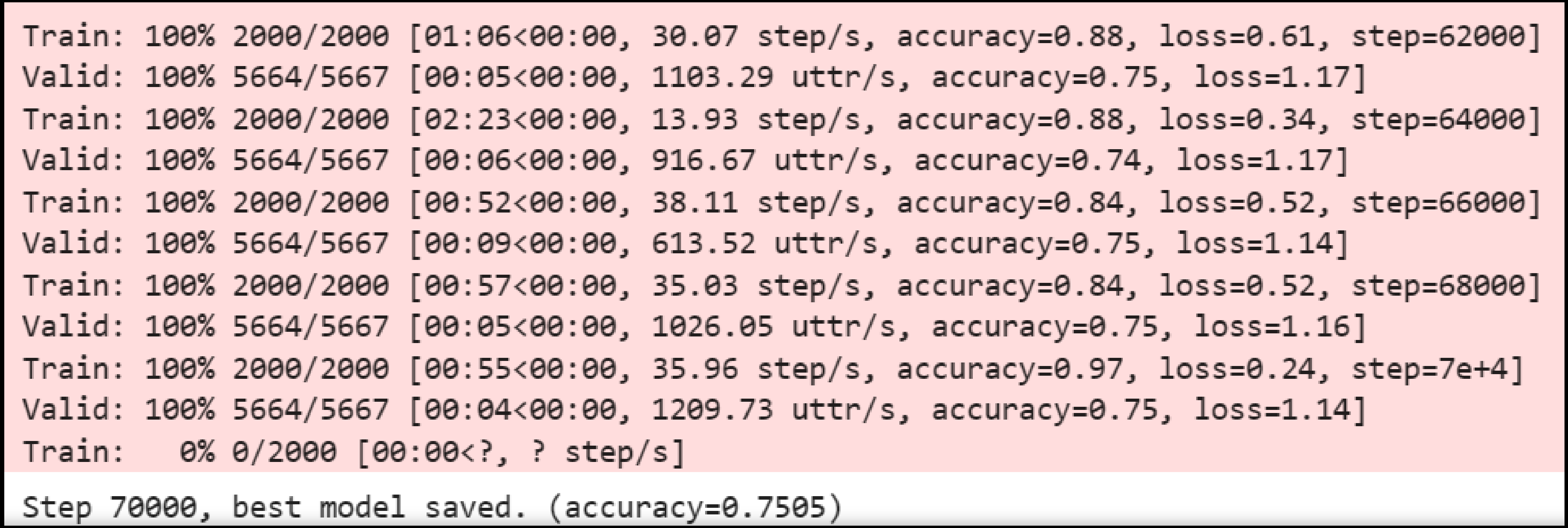


```python
class Classifier(nn.Module):
    def __init__(self, d_model=96, n_spks=600, dropout=0.1):
        super(Classifier, self).__init__()
        self.prenet = nn.Linear(40, d_model)
        self.positional_encoding = PositionalEncoding(d_model)
        self.conformer_encoder = ConformerEncoder(num_layers=3, d_model=d_model, nhead=4, dropout=dropout)
        self.pred_layer = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.ReLU(),
            nn.Linear(d_model, n_spks),
        )

    def forward(self, mels):
        out = self.prenet(mels)
        out = out.permute(1, 0, 2)
        out = self.positional_encoding(out)
        out = self.conformer_encoder(out)
        out = out.transpose(0, 1)
        stats = out.mean(dim=1)
        out = self.pred_layer(stats)
        return out
```



```
The parameter size of encoder block is 490.776k
[Info]: Use cuda now!
[Info]: Finish loading data!
[Info]: Finish creating model!
```

Task 2 Result

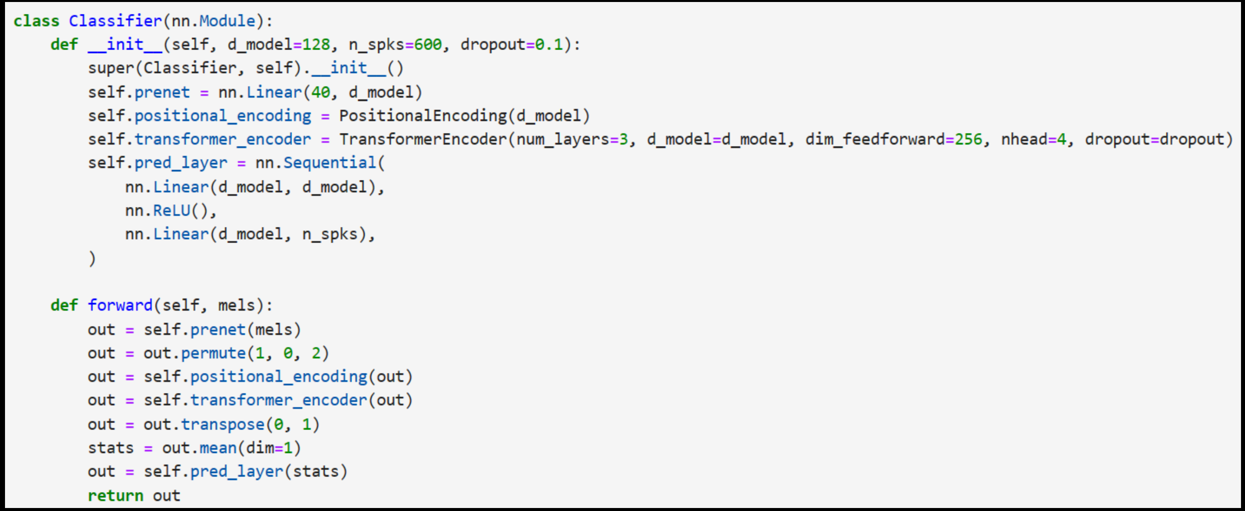


```
Train: 100% 2000/2000 [01:06<00:00, 30.07 step/s, accuracy=0.88, loss=0.61, step=62000]
Valid: 100% 5664/5667 [00:05<00:00, 1103.29 uttr/s, accuracy=0.75, loss=1.17]
Train: 100% 2000/2000 [02:23<00:00, 13.93 step/s, accuracy=0.88, loss=0.34, step=64000]
Valid: 100% 5664/5667 [00:06<00:00, 916.67 uttr/s, accuracy=0.74, loss=1.17]
Train: 100% 2000/2000 [00:52<00:00, 38.11 step/s, accuracy=0.84, loss=0.52, step=66000]
Valid: 100% 5664/5667 [00:09<00:00, 613.52 uttr/s, accuracy=0.75, loss=1.14]
Train: 100% 2000/2000 [00:57<00:00, 35.03 step/s, accuracy=0.84, loss=0.52, step=68000]
Valid: 100% 5664/5667 [00:05<00:00, 1026.05 uttr/s, accuracy=0.75, loss=1.16]
Train: 100% 2000/2000 [00:55<00:00, 35.96 step/s, accuracy=0.97, loss=0.24, step=7e+4]
Valid: 100% 5664/5667 [00:04<00:00, 1209.73 uttr/s, accuracy=0.75, loss=1.14]
Train:   0% 0/2000 [00:00<?, ? step/s]
Step 70000, best model saved. (accuracy=0.7505)
```

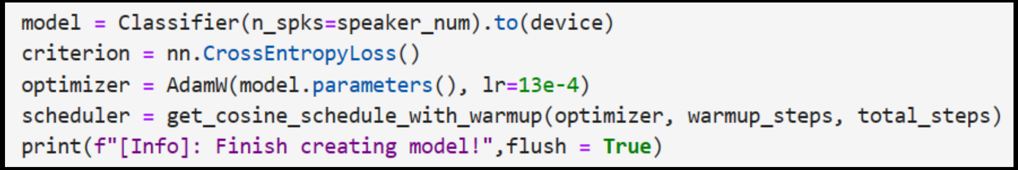
Improvement of Task 1

○ 參數微調

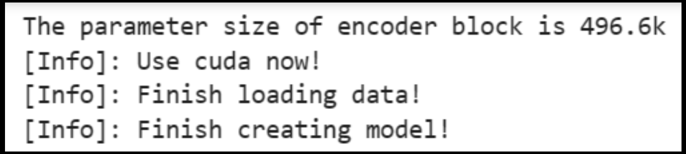
參數方面我一直針對模型隐藏层的维度和 α (學習率) 做調整。當我調整模型隐藏层的维度和前饋神經網路隱藏層維度的時候，我發現encoder block的參數大小都會受到影響，尤其是前饋神經網路，每次倍數增加的時候，encoder block的參數就隨之增加100k。因此我只對模型隱藏層做更改，我把隐藏层的维度設置爲128，最終encoder block的參數大小壓在496.6k(<500k)。至於 α，一開始以0.1、0.01、0.001做測試，發現介於0.0008~0.002的 α 產生的精度比較高，然後再從中以0.0001的增幅做測試，最終 α 為0.0013的精度最好。我的Final Accuracy為73.64%。



```
class Classifier(nn.Module):
    def __init__(self, d_model=128, n_spks=600, dropout=0.1):
        super(Classifier, self).__init__()
        self.prenet = nn.Linear(40, d_model)
        self.positional_encoding = PositionalEncoding(d_model)
        self.transformer_encoder = TransformerEncoder(num_layers=3, d_model=d_model, dim_feedforward=256, nhead=4, dropout=dropout)
        self.pred_layer = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.ReLU(),
            nn.Linear(d_model, n_spks),
        )

    def forward(self, mels):
        out = self.prenet(mels)
        out = out.permute(1, 0, 2)
        out = self.positional_encoding(out)
        out = self.transformer_encoder(out)
        out = out.transpose(0, 1)
        stats = out.mean(dim=1)
        out = self.pred_layer(stats)
        return out
```

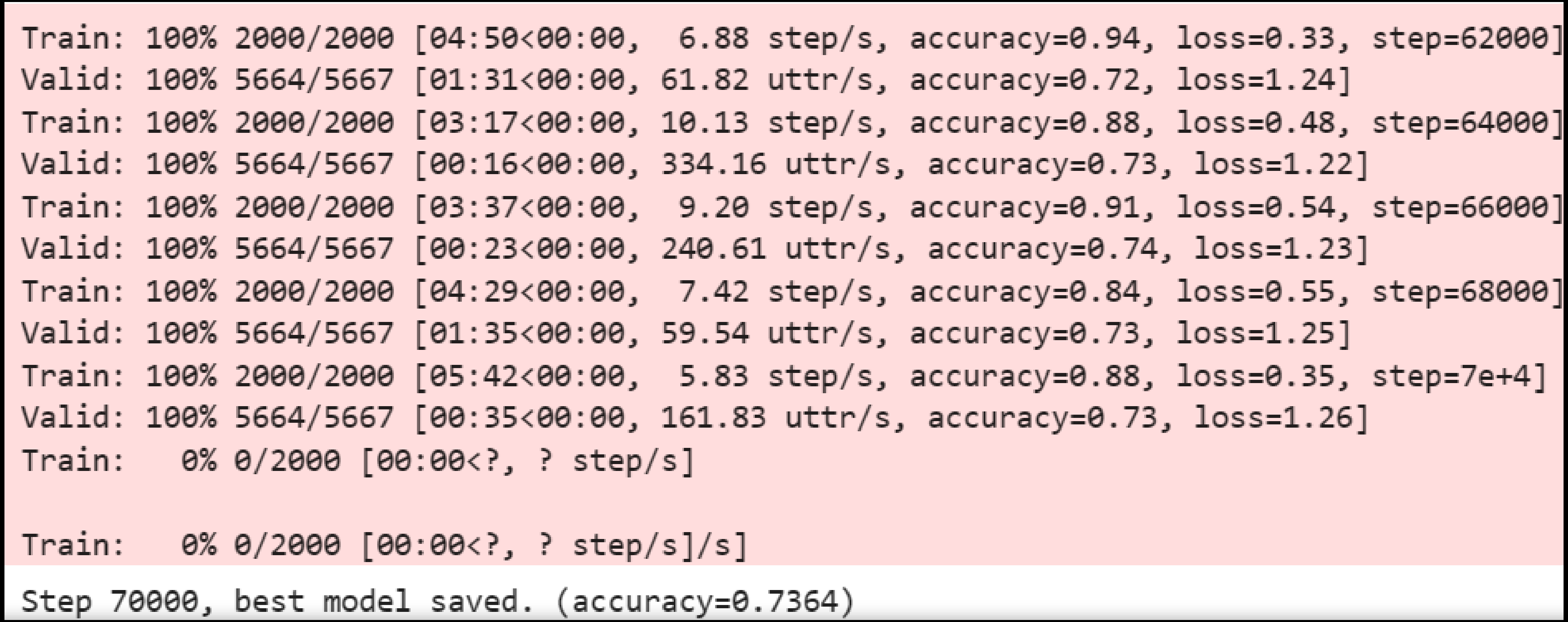


```
model = Classifier(n_spks=speaker_num).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = AdamW(model.parameters(), lr=13e-4)
scheduler = get_cosine_schedule_with_warmup(optimizer, warmup_steps, total_steps)
print(f"[Info]: Finish creating model!",flush = True)
```



```
The parameter size of encoder block is 496.6k
[Info]: Use cuda now!
[Info]: Finish loading data!
[Info]: Finish creating model!
```

Task 1 Final Result



```
Train: 100% 2000/2000 [04:50<00:00,  6.88 step/s, accuracy=0.94, loss=0.33, step=62000]
Valid: 100% 5664/5667 [01:31<00:00, 61.82 uttr/s, accuracy=0.72, loss=1.24]
Train: 100% 2000/2000 [03:17<00:00, 10.13 step/s, accuracy=0.88, loss=0.48, step=64000]
Valid: 100% 5664/5667 [00:16<00:00, 334.16 uttr/s, accuracy=0.73, loss=1.22]
Train: 100% 2000/2000 [03:37<00:00,  9.20 step/s, accuracy=0.91, loss=0.54, step=66000]
Valid: 100% 5664/5667 [00:23<00:00, 240.61 uttr/s, accuracy=0.74, loss=1.23]
Train: 100% 2000/2000 [04:29<00:00,  7.42 step/s, accuracy=0.84, loss=0.55, step=68000]
Valid: 100% 5664/5667 [01:35<00:00, 59.54 uttr/s, accuracy=0.73, loss=1.25]
Train: 100% 2000/2000 [05:42<00:00,  5.83 step/s, accuracy=0.88, loss=0.35, step=7e+4]
Valid: 100% 5664/5667 [00:35<00:00, 161.83 uttr/s, accuracy=0.73, loss=1.26]
Train:   0% 0/2000 [00:00<?, ? step/s]

Train:   0% 0/2000 [00:00<?, ? step/s]/s]
Step 70000, best model saved. (accuracy=0.7364)
```