Multimedia Content Analysis Homework 5

Music Genre Classification

WEI-CHIH CHANG Q36111281

May 15, 2023

Abstract

本文透過 Vision Transformer 技術實作音樂風格辨識的任務。透過短時傅立葉轉換和 window slide 技術，將聲音訊號轉換成時頻圖，再將時頻圖根據 step 拆分成訓練集，最後將訓練集作為視覺資料輸入至 Transformer model，將編碼後的資料透過解碼器匹配至對應的音樂風格。

1. Related Work

隨著 ChatGPT 的爆紅，基於 self-attention 技術的 Transformer model 逐漸從人工智慧相關從業者或是學者的圈子，開始進入大眾的視野。而 Transformer model 首次被提到是在 2017 年，由 Google Brain 團隊的 Vaswani, Shazeer 等人所發表的 "Attention is all you need." 中。文章中提到「Transformer model 被用於機器翻譯時基本包含一個用於讀取輸入句子並生成其表示的編碼器。接著解碼再參考編碼器生成的結果同時逐字生成輸出的句子」[1]。

如同上述所提到的，這次會使用 Transformer model 來完成音樂風格辨識的任務是因為，一開始我認為語音辨識或是音樂風格辨識等任務其實和機器翻譯一樣，都是 Seq2Seq 的任務，在數學上都可以被寫成相似的數學模型，因此最一開始我是選用原始論文的 Transformer model 架構進行修改，也就是將音樂的波型直接視為文字的 token 處理。然而過程中我發現音樂波型的語意資料量遠小於文字的語意，也就是說音檔中可能要 4096 個點才相當於中文中「我」的語意的資料量，同時聲音訊號中代表一個 token 的長度也都大不相同。除此之外，要建立能直接處理聲音訊號的 Transformer model 需要的參數量十分龐大，光是編碼後和映射層連接之間的參數，其參數量就相當於簡單 CNN 模型所使用到的參數數量了。

為了解決在有限資源的情況下，Transformer model 無法直接處理聲音訊號的問題，我參考 Alexey, Lucas 等人將 Transformer model 應用在電腦視覺的論文[2]，以及 Yuan 等人將時頻圖做為二維資料應用至 Transformer model 中的方法[3]，成功在有限資源的筆記型電腦中實現小型的 Transformer model 並進行訓練。在訓練集和驗證集比例為 8:2 的情境下，模型在訓練 30 個 epoch 時驗證集精確度為 0.232。

2. Transformer

在聲音訊號的辨識任務中，其中一項是根據聲音訊號進行情感分析，相關的技術包含音訊處理和自然語言處理。而自 2020 年以後的這幾年，處理自然語言常見的方法是使用應用了 self-attention 技術的 Transformer model。

Transformer model 的模型架構是基於 Seq2Seq 和 self-attention 技術設計，不同於過去在自然語言任務中常見的 Recurrent Neural Network，像是 Long short-term memory 或是 Gate Recurrent Unit 等針對包含時間訊息的時序型資料設計的模型[4][5]。Transformer model 與 Recurrent Neural Network 不同，Transformer model 是可以進行平行運算的，這是因為 Transformer model 使用 self-attention 取代 recurrent 的過程，因此在 loss 傳遞的過程不需要依靠前一級的資料傳遞。

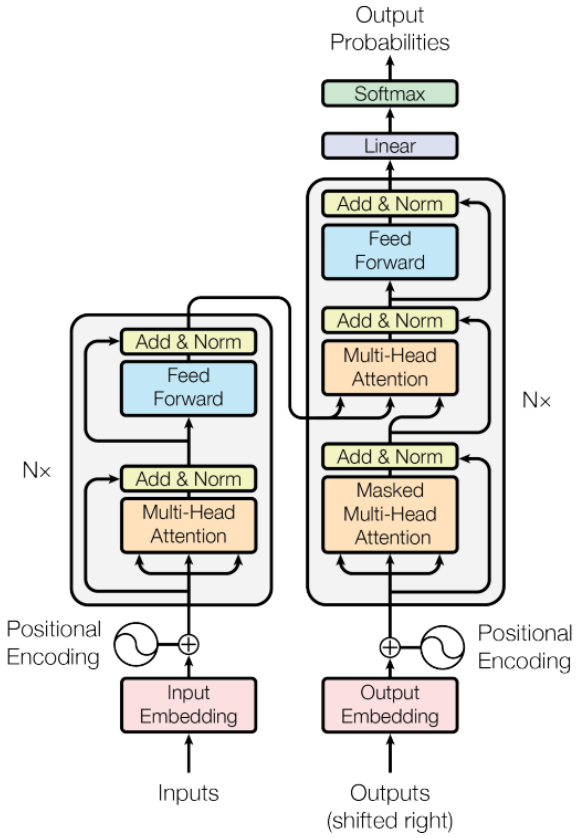


Figure 1. Transformer model 最原始的架構[1]

在原始的 Transformer model 架構中，編碼器和解碼器分別由六個區塊所組成，而在音樂風格辨識的任務中，因為是在筆記型電腦中訓練模型，並沒有額外的 GPU 輔助模型的訓練，因此將編碼器的區塊增加為八個，然後將並行的資料輸入縮減為四個，這樣的改動是為了符合 CPU 的計算模式，同時解碼器的部分則因應任務需求改為 Multilayer perceptron。原始論文中的 Word2Vec 在本次 project 中被改為 Img2Vec，並且嵌入的維度從 512 降至 128。

3. Vision Transformer

Transformer model 作為通用的深度學習模型，除了應用在自然語言的處理任務上，也能應用在電腦視覺的任務當中。Vision Transformer 作為第一個將視覺資料應用到 Transformer model 的模型，它的策略是將圖片拆成固定大小的 patch，接著對每個 patch 進行線性嵌入並添加位置資訊，並將得到的向量序列送入一個標準的 Transformer 編碼器。而在原始的 Vision Transformer 論文中，作者有在 Patch Embedding 的序列中加入一個額外的可學習的「分類標記」到序列中，但在我們的 project 中捨棄這個設計，因為在原作者的論文中有提到該方法對效能的提升程度有待商榷[2]。

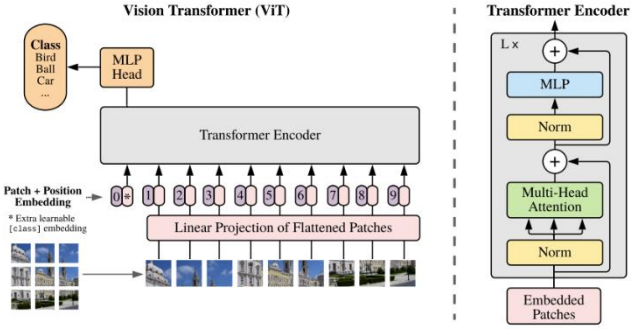


Figure 2. Vision Transformer 的設計架構[2]

為了得到 Vision Transformer 的輸入圖片，我們設計 window size 為 256；slide size 為 1 的短時傅立葉轉換窗口，透過上述操作我們將得到每一個三十秒音樂的時頻圖。接著我們將每個時頻圖進行 batch 的拆分，為了在檔案大小、Vision Transformer 參數數量以及訓練集資料量三者中做取捨，我們最終選擇 512 為 batch size, 因為在取樣率為 22050 的音檔中，這樣的 batch size 相當於五秒的音樂片段，對一般人而言，這樣的長度是有足夠的特徵進行音樂類別的辨識的，同時又不會造成訓練集資料量過大和模型參數過大。

在 patch size 的部分，原始論文以及其他論文大多都採用正方形的 patch size。在 image size 為[128, 32]時，由於時頻譜的特性，我選擇[16, 4]的 patch size 設計，不使用根據時間切分的[2, 32]是因為有論文提到：因為 Vision Transformer 會對 Patch 進行 Embedding，如果使用[2, 32]甚至是[1, 64]的 patch size 就相當於浪費了 Embedding 將時間資料嵌入到 patch 的作用了。

4. Train

在訓練的過程中，Vision Transformer 需要較小的 learning rate 來達到較好的訓練效果。同時，我在 batch 輸入模型後對 batch 進行 resize 用以減少訓練時間，在 epochs 小於 10 時，模型幾乎看不到訓練成效，直至 epochs 大於 20 後，才會有明顯的訓練成效，但由於硬體資源和時間限制，最終我選擇 epochs 為 30 來驗證模型的訓練結果。

Cross validation 的部分由於訓練時間與硬體資源的的緣故，我只進行兩輪的 cross 便進行報告的撰寫。

5. Performance

從 Transformer model 的架構中可以得知，Transformer model 作為泛用模型，它的架構雖然能將聽覺、視覺和語言訊息整合在一起，但在非並行架構的處理器上，它的訓練難度遠高於 CNN 或是 RNN 等模型，同時 Transformer model 在小型資料集上完全占不到便宜，因為 Transformer model 需要在更大的資料集上學習高品質的中間值表示[6]。

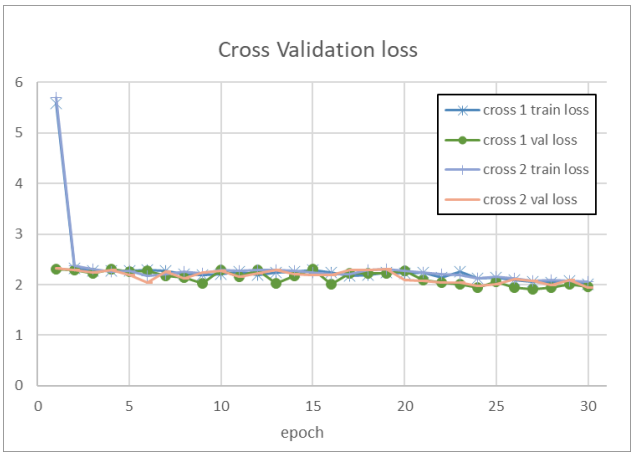


Figure 3. 兩次 cross validation。loss 與 epoch 之間的關係

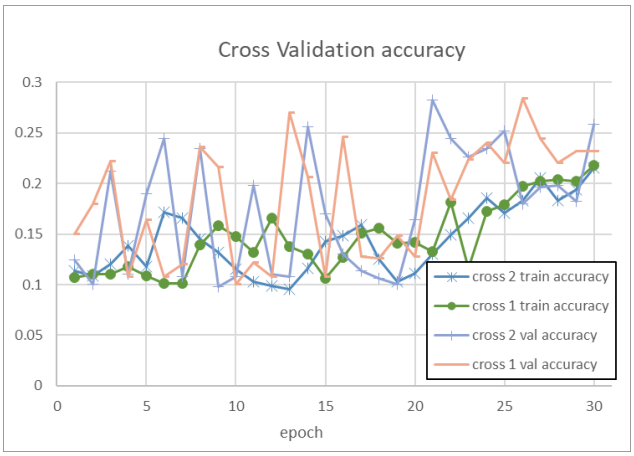


Figure 4. 兩次 cross validation。accuracy 與 epoch 之間的關係

從圖三以及圖四可以發現，雖然在 epoch=30 時，可以注意到訓練的過程中 loss 和 accuracy 分別有下降和上升的趨勢，但數值上仍舊很浮動。Loss 在剛開始訓練時才有大幅的下降，但在整個過程中 loss 的值都在 2 上下徘徊。Accuracy 的部分則可以發現兩次 cross validation 的 train accuracy 都呈現穩定上升的趨勢，可以說明模型有在漸漸地學習如何將「注意力」正確的擺在 train data 的正確位置上，然而在 test data 上，模型的表現就叫為浮動，這也說明模型尚未學習到較好的中間曾表示。

6. Conclusion

從這次作業可以發現在硬體資源及時間資源不足的情況下，Transformer model 完全占不到任何便宜，不僅訓練時間較常見應用於音訊的模型，如 RNN 或是 LSTM 長，同時需要的訓練資料也比前面提到的時序模型高上數倍。

不過 Transformer model 中應用的 Autoencoder 和 Self-attention 架構非常值得學習，例如 Autoencoder 應用在分類、特徵提取或是特徵解耦的任務，或是 Self-attention 應用在 CNN 上，藉此達到快速從訓練集中學習到局部和全域的特徵的目的。不僅如此 Transformer model 對於輸入以及輸出的泛用程度，適合作為不同模型之間的橋接模型。

附錄 1. Vision Transformer Music Genre Classification Model Architecture

參考文獻

[1] Vaswani Ashish, Shazeer Noam, Parmar Niki, Uszkoreit Jakob, Jones Llion, Gomez Aidan N., Kaiser Łukasz, and Polosukhin Illia. 2017. Attention is all you need. In Advances in Neural Information Processing Systems. 5998–6008.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR, 2021.

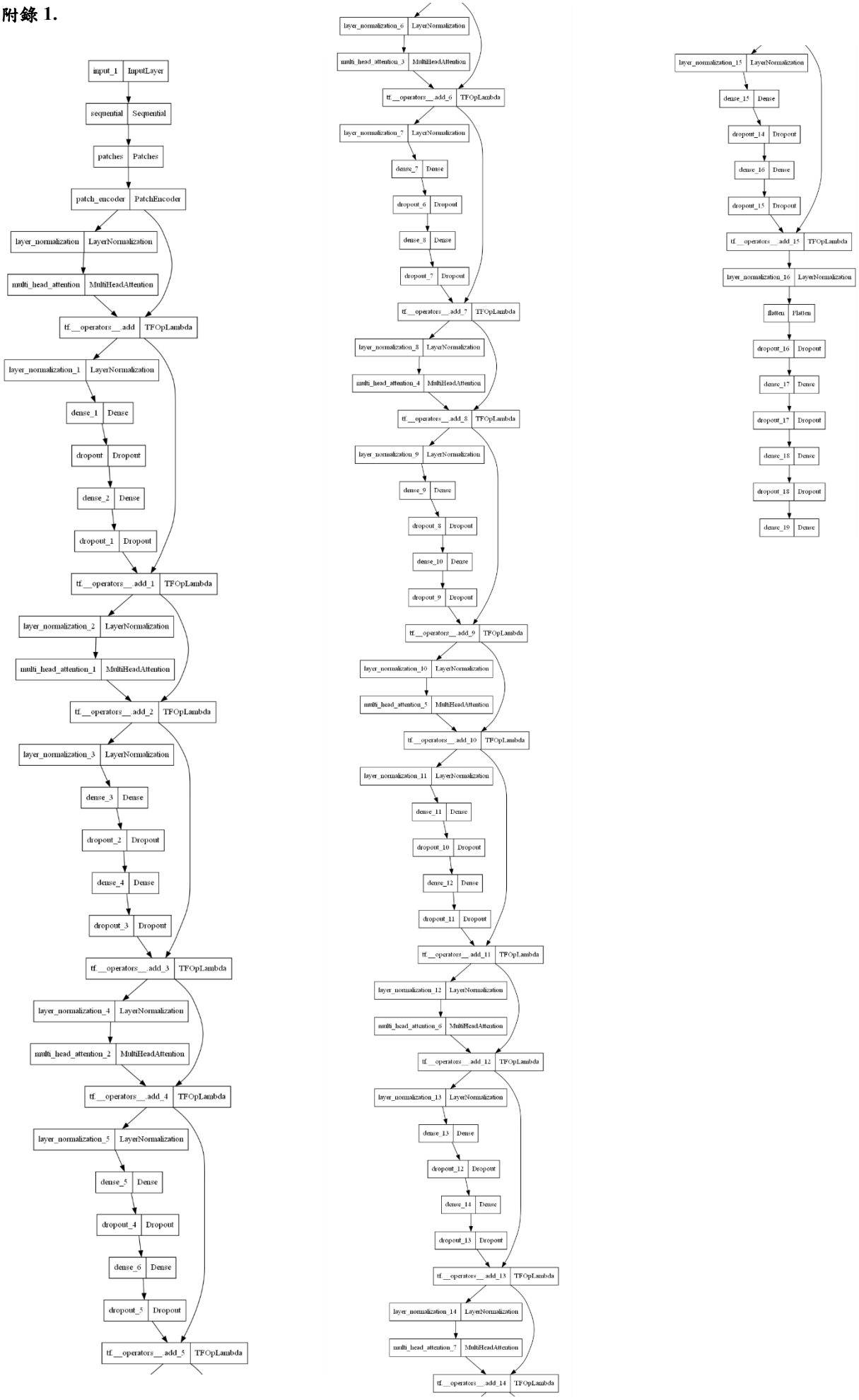
[3] Yuan Gong, Yu-An Chung, and James Glass. AST: audio spectrogram transformer. arXiv preprint arXiv:2104.01778, 2021.

[4] Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems 3104–3112 (NIPS, 2014).

[5] Kyunghyun Cho, Bart van Merrienboer, C¸ aglar Gulc¸ehre, ¨ Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[6] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? arXiv preprint arXiv:2108.08810, 2021.

附錄 1.


input_1 InputLayer
sequential Sequential
patches Patches
patch_encoder PatchEncoder
layer_normalization LayerNormalization
multi_head_attention MultiHeadAttention
tf.__operators__.add TFOpLambda
layer_normalization_1 LayerNormalization
dense_1 Dense
dropout Dropout
dense_2 Dense
dropout_1 Dropout
tf.__operators__.add_1 TFOpLambda
layer_normalization_2 LayerNormalization
multi_head_attention_1 MultiHeadAttention
tf.__operators__.add_2 TFOpLambda
layer_normalization_3 LayerNormalization
dense_3 Dense
dropout_2 Dropout
dense_4 Dense
dropout_3 Dropout
tf.__operators__.add_3 TFOpLambda
layer_normalization_4 LayerNormalization
multi_head_attention_2 MultiHeadAttention
tf.__operators__.add_4 TFOpLambda
layer_normalization_5 LayerNormalization
dense_5 Dense
dropout_4 Dropout
dense_6 Dense
dropout_5 Dropout
tf.__operators__.add_5 TFOpLambda
layer_normalization_6 LayerNormalization
multi_head_attention_3 MultiHeadAttention
tf.__operators__.add_6 TFOpLambda
layer_normalization_7 LayerNormalization
dense_7 Dense
dropout_6 Dropout
dense_8 Dense
dropout_7 Dropout
tf.__operators__.add_7 TFOpLambda
layer_normalization_8 LayerNormalization
multi_head_attention_4 MultiHeadAttention
tf.__operators__.add_8 TFOpLambda
layer_normalization_9 LayerNormalization
dense_9 Dense
dropout_8 Dropout
dense_10 Dense
dropout_9 Dropout
tf.__operators__.add_9 TFOpLambda
layer_normalization_10 LayerNormalization
multi_head_attention_5 MultiHeadAttention
tf.__operators__.add_10 TFOpLambda
layer_normalization_11 LayerNormalization
dense_11 Dense
dropout_10 Dropout
dense_12 Dense
dropout_11 Dropout
tf.__operators__.add_11 TFOpLambda
layer_normalization_12 LayerNormalization
multi_head_attention_6 MultiHeadAttention
tf.__operators__.add_12 TFOpLambda
layer_normalization_13 LayerNormalization
dense_13 Dense
dropout_12 Dropout
dense_14 Dense
dropout_13 Dropout
tf.__operators__.add_13 TFOpLambda
layer_normalization_14 LayerNormalization
multi_head_attention_7 MultiHeadAttention
tf.__operators__.add_14 TFOpLambda
layer_normalization_15 LayerNormalization
dense_15 Dense
dropout_14 Dropout
dense_16 Dense
dropout_15 Dropout
tf.__operators__.add_15 TFOpLambda
layer_normalization_16 LayerNormalization
flatten Flatten
dropout_16 Dropout
dense_17 Dense
dropout_17 Dropout
dense_18 Dense
dropout_18 Dropout
dense_19 Dense