毫米波雷達AI創意競賽

書面報告

隊伍：天上太陽紅通通

主題：Otamatone實作

成員：張簡雲翔、陳奕羶、劉德權、王駿彥

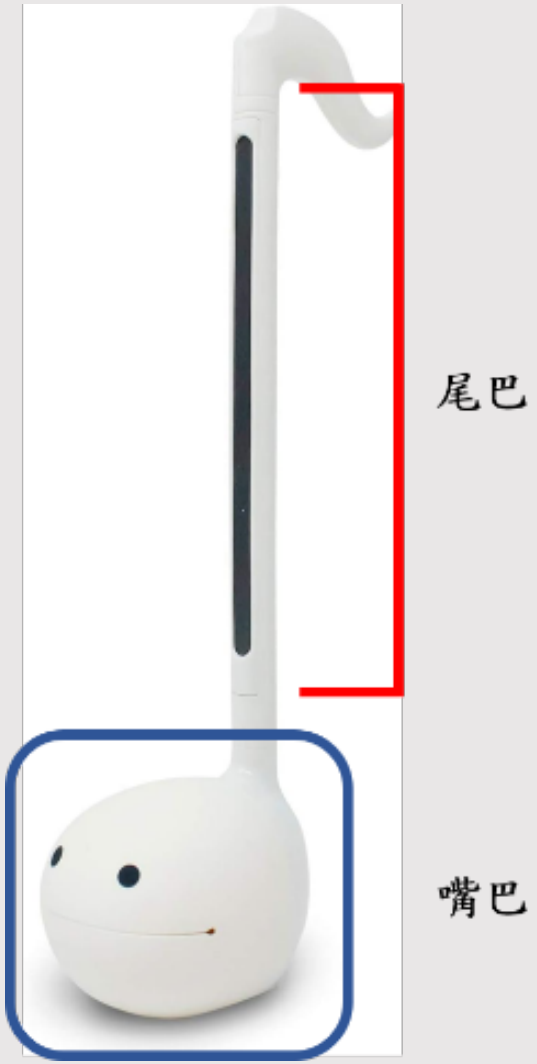
Outline

- Introduction
- Background
- System Structure
- Implementation and Difficulty
- Conclusion
- Reference
- Appendix

Introduction

Introduction

- Otamatone
  - 分成兩部分：嘴巴跟尾巴
  - 上到下是高音到低音
  - 演奏者不用忍受按弦的痛苦
  - 可以製造出滑音等效果
- Goal
  - 透過毫米波雷達模擬尾巴的滑軌
  - 簡化不做滑音的部分
  - 透過位置去判斷按下的音調
  - 透過電腦的軟體播放現在的音調



Introduction

From：https://www.youtube.com/watch?v=cC5ZJ8Y4OuQ

Background

Background

- K60168A Dongle
  - 一個透過毫米波辨識前方物體移動的模組
  - 可以辨識出方位、距離、速度
  - 透過官方提供的軟體可以收集手勢影像
  - 透過官方提供的原始碼可以讓即時收集資料
  - 作為本次辨別手勢位置的套件
  - 需要用分辨影像的模型作為主模型
- 影像辨識模型
  - ConvLSTM
  - R(2+1)D CNN
  - ViViT

Video Process Model(1/2)

- ConvLSTM
  - 2015年提出的一個模型[1]
  - 是基於LSTM算法的一種改良
  - 為了讓LSTM對於影像處裡的效果更好，將部分原先的矩陣乘法部分改為捲積運算
- R(2+1)D CNN
  - 2017 Meta提出的一個模型[2]
  - 一般CNN處裡影片都是用Conv3D
  - 類似Depthwise Separable Convolution
  - 將原先Conv3D的做法拆為Spatio部分跟Temporal部分((2+1D) convolution)

Video Process Model(2/2)

- ViViT
  - 2021年由Google提出的一個模型[3]
  - 主要希望利用純Transfomer做影像分類
  - 論文中提出了四種不同的架構
    - 第一種是單純的Transformer
    - 後三種是在建構時間與空間的Transformer

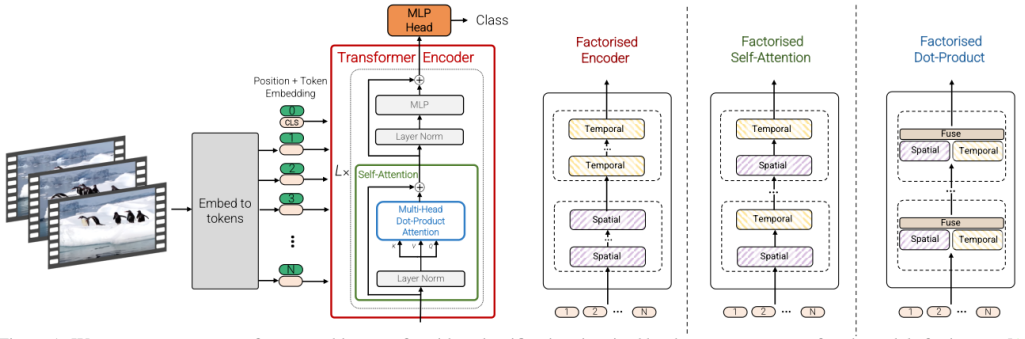


Figure 1: We propose a pure-transformer architecture for video classification, inspired by the recent success of such models for images [15]. To effectively process a large number of spatio-temporal tokens, we develop several model variants which factorise different components of the transformer encoder over the spatial- and temporal-dimensions. As shown on the right, these factorisations correspond to different attention patterns over space and time.

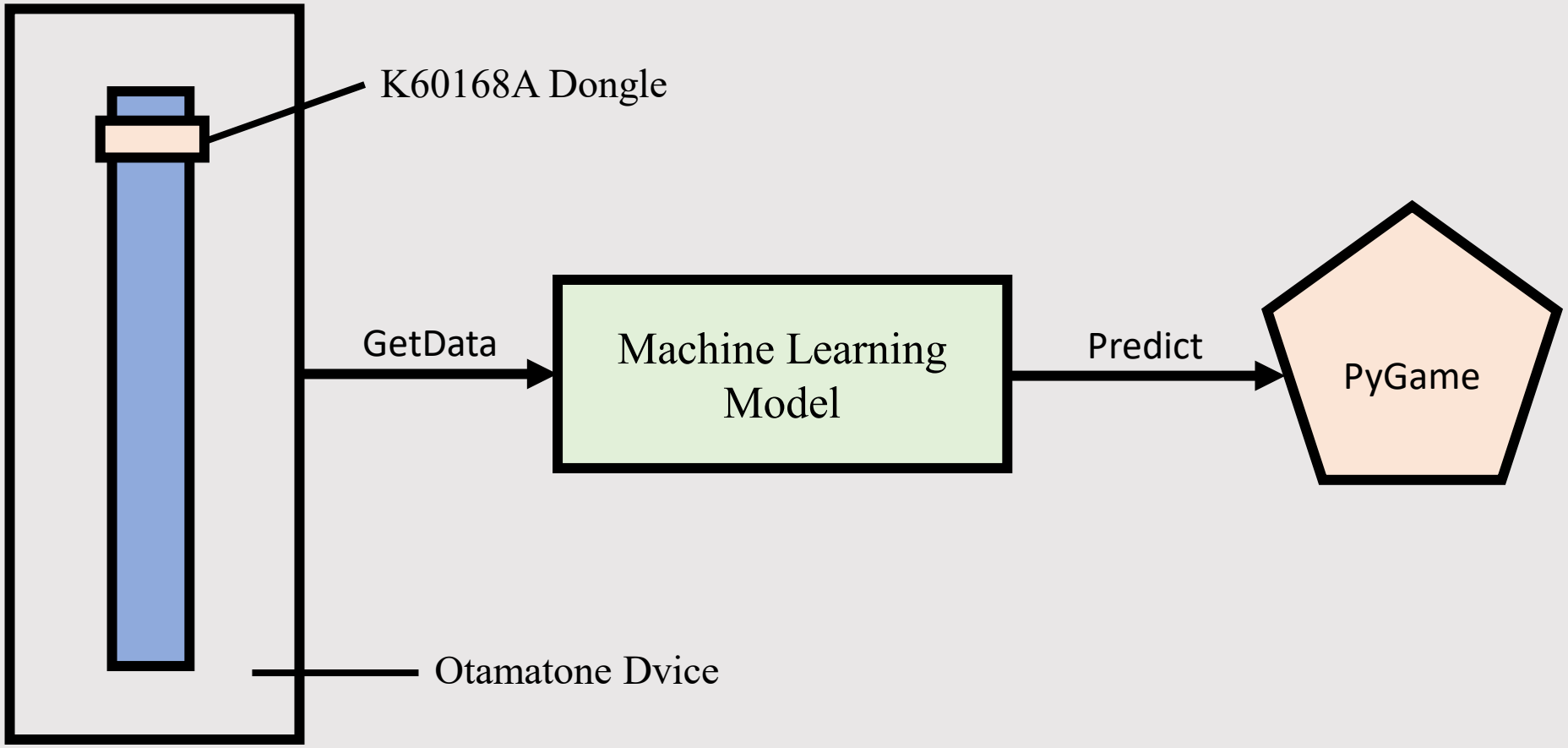
  - 考量算力與資料量較少本次沒有選用這個模型
    - 僅提供一種參考用的模型，畢竟這個還算滿有名的一個模型

System Structure

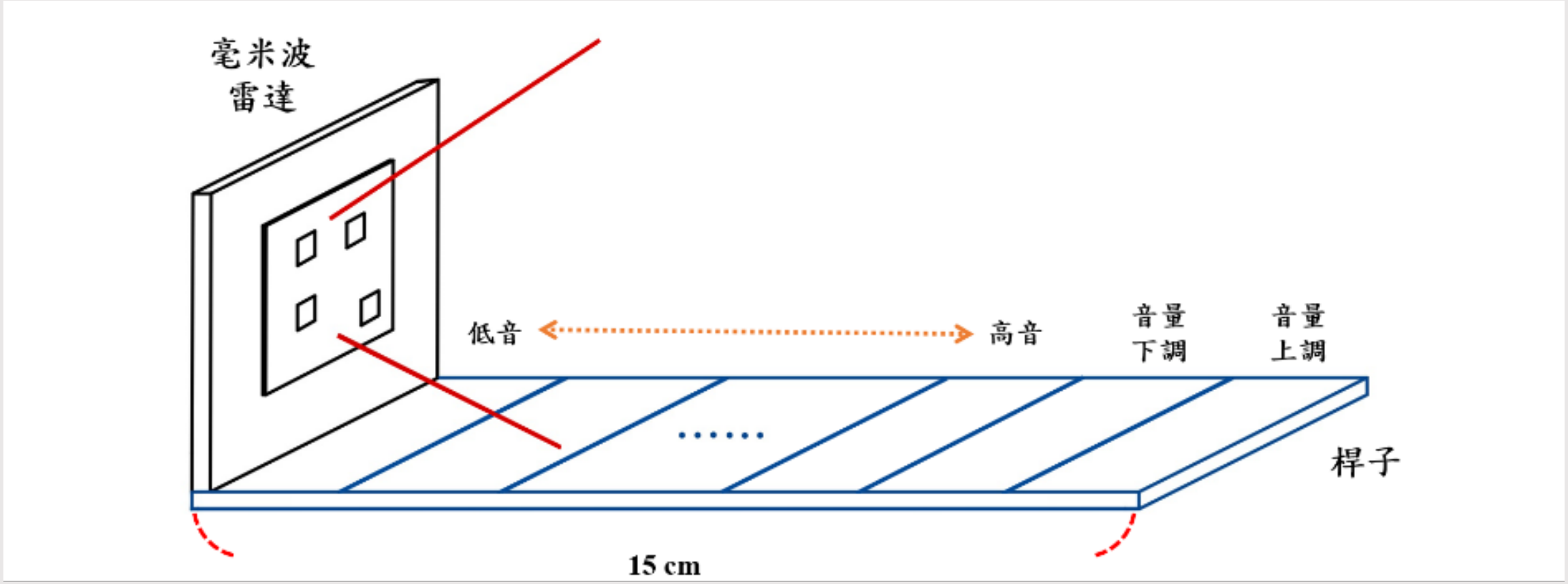
Overview(1/2)

- 將K60168A Dongle固定在一根長度約15~20公分的桿子上，並透過傳輸線連接到電腦
- 使用時會隨時收集資料並傳輸給電腦最後輸入到模型中
- 模型會根據收集到的資料進行預測
- 預測完後，將結果輸出到PyGame中並調用Windows內建的合成器，並執行相應的動作
- 原始設計想用Sonic Pi來完成放出聲音，但因為Sonic Pi的合成器聲音太難聽，改用PyGame來調用Windows內建的合成器
- 並在電腦上可以選擇現在要演奏的樂器聲音以及音域

Overview(2/2)


K60168A Dongle
GetData
Machine Learning Model
Predict
PyGame
Otamatone Dvice

Device Concept

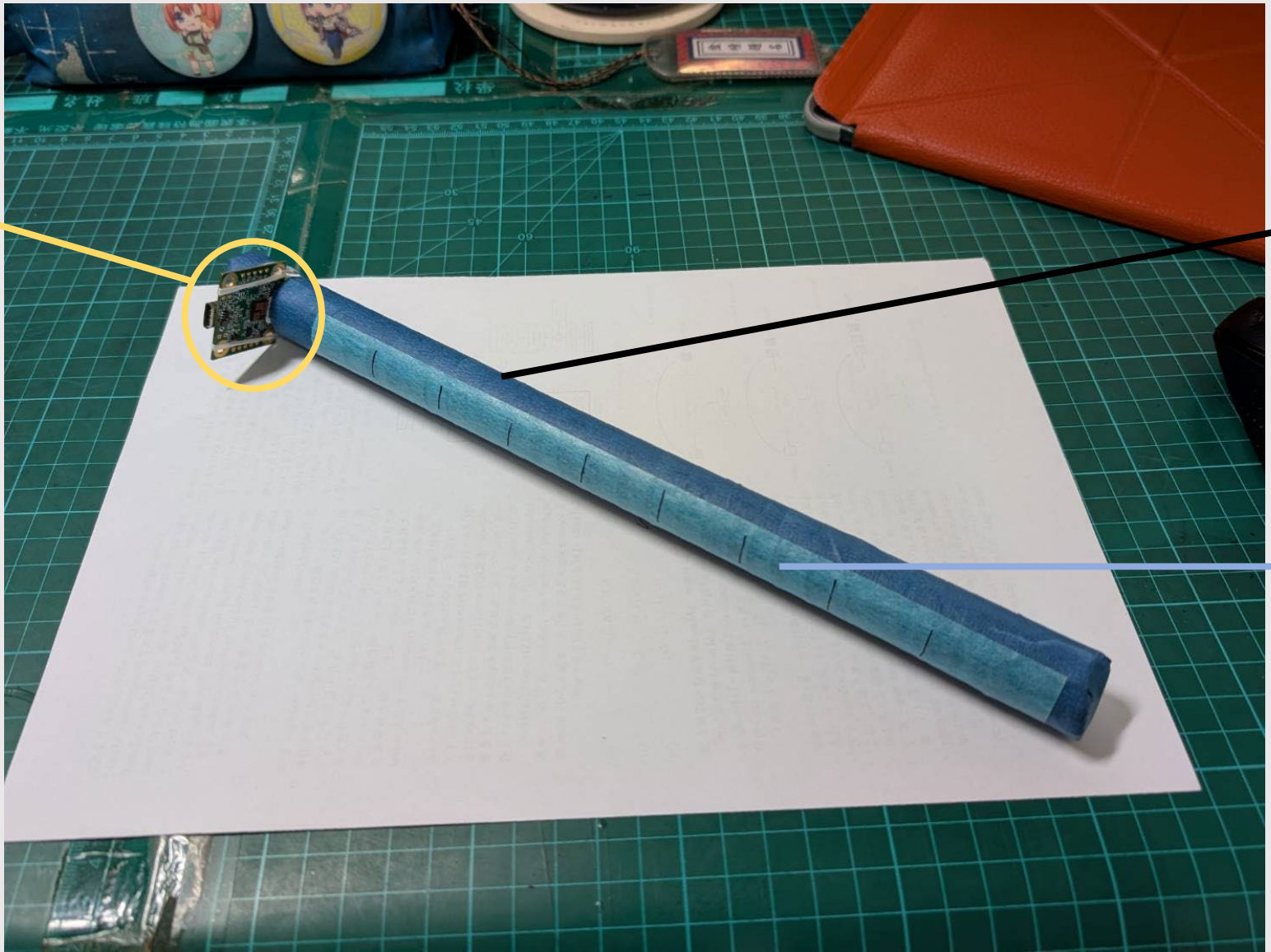


Implementation and Difficultly

Device Implementation(1/3)

- 收集一根長度約20公分的桿子
  - 因為我們是窮學生，所以去外面撿了一根短短的竹子回來
  - 之後先把竹子中較為筆直的地方保留其他部分鋸掉
  - 約莫最後得到一根長30公分左右的竹子
  - 用膠帶捆住竹子外部避免竹子刮到手
  - 至此獲得一根好用的桿子
- 將桿子前半部約5公分處鋸一個開口，作為Dongle的固定位置
- 之後拿棉繩纏繞固定板子並綁住
- Dongle前方黏一個約長25公分的膠帶作為滑軌，並標記位置
  - 原本規劃是15公分，後面實作發現過於困難，所以增加10公分

Device Implementation(2/3)



Device Implementation(3/3)

Otamatone Device
拿在手上的感覺

Data Collection(1/2)

- 透過官方提供的Collect_RDI軟體收集已經過Fourier Transform後的資料
- 收集方式
  - 透過軟體裡面的pre define的mode，設定固定frame是動作標記，並設定固定間隔
  - 因為總frame太大會導致資料前處理過久，所以設定為190
  - 每次動作12 frames
  - 每次動作間隔數個frames
- 收集畫面範例（右下角為人操作otamatone的畫面）

Data Collection(2/2)

Data Splitting

- 我們合計收集307筆的資料，作為我們資料集，每個資料裡面都有10種label，每數個frame的label會變換一次，label如下：

  0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0

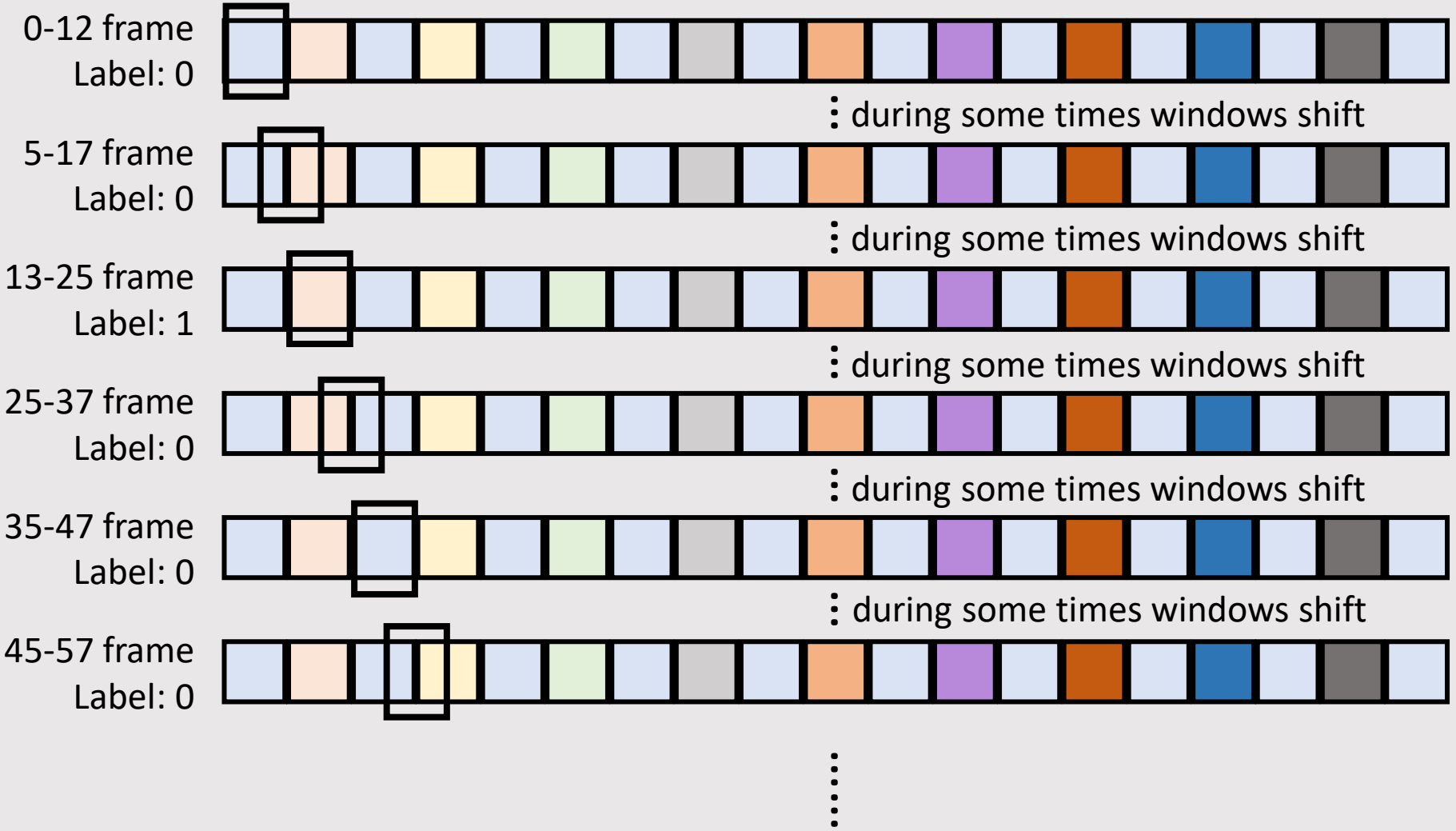
- 我們資料集切成三份作為training set、validation set、test set，比例如下：
  - Training set ：192（64%）
  - Validation set ： 49（16%）
  - Test set ： 62（20%）

Data Preprocessing(1/2)

- 原始資料：190 frame rate 並有多個label，分別如下：

  0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0
- Label 0 代表沒有動作，label 1 ~ 7由低到高分別為Do Re Mi Fa So La Si，label 8代表聲音上調，label 9代表聲音下調。
- 為了後續訓練分辨，所以需要把各個label的資料從單一資料分割出來
- 我們用移動window的方式一個一個去掃描切割，window size 設為12，當window中所有label皆為一樣，才標記為該label，其他的則視為label 0，為避免資料產生極大的偏態，所以訓練時只有隨機選取與其他label一樣量的label 0

Data Preprocessing(2/2)



Model Overview(1/3)

- CNN Model
  - Version 1
    - Total params: 969,098
    - Trainable params: 969,098
    - Non-trainable params: 0
  - Version 2
    - Total params: 1,002,122
    - Trainable params: 1,002,122
    - Non-trainable params: 0

Model Overview(2/3)

- ConvLSTM Model
  - Version 3
    - Total params: 278,730
    - Trainable params: 278,538
    - Non-trainable params: 192
  - Version 4
    - Total params: 782,026
    - Trainable params: 781,834
    - Non-trainable params: 192
  - Version 5
    - Total params: 782,026
    - Trainable params: 781,834
    - Non-trainable params: 192
  - Version 6
    - Total params: 782,026
    - Trainable params: 781,834
    - Non-trainable params: 192

相同的模型架構
僅有optimizer變換以及資料是否有shuffle
進行測試

Model Overview(3/3)

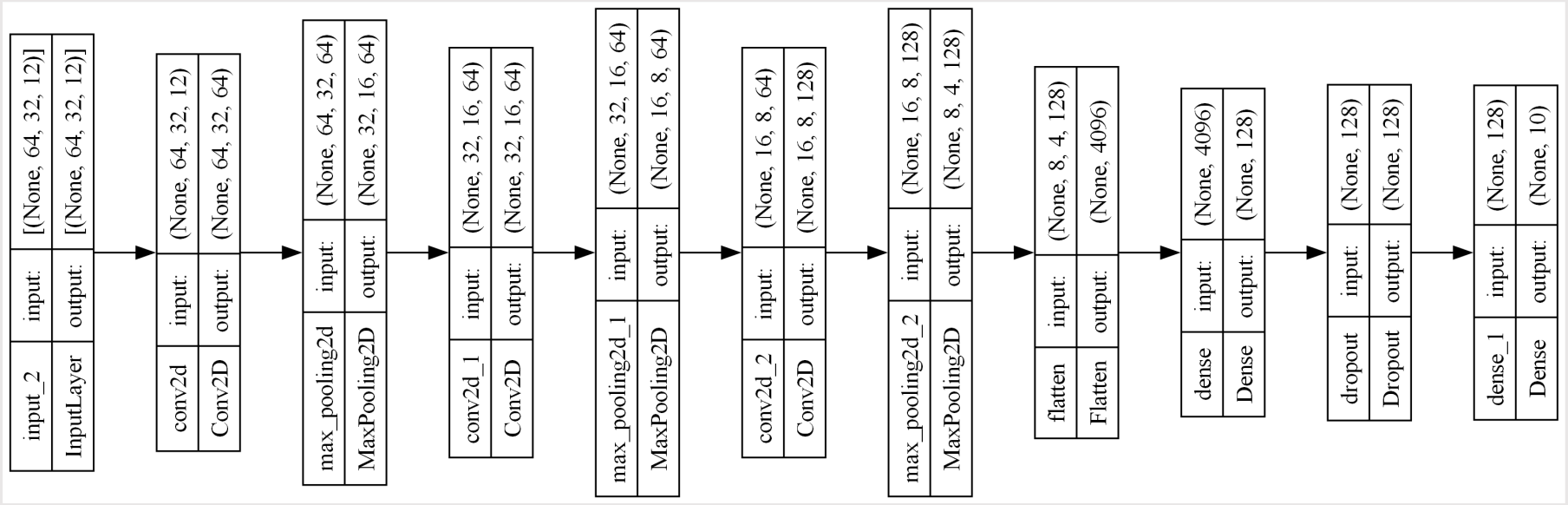
- R(2+1)D Model
  - Version 7
    - Total params: 450,154
    - Trainable params: 450,122
    - Non-trainable params: 32
  - Version 8
    - Total params: 431,434
    - Trainable params: 431,370
    - Non-trainable params: 64
  - Version 9
    - Total params: 126,250
    - Trainable params: 126,218
    - Non-trainable params: 32

Model Version 1(1/3)

- 因為原先想說我們的手勢並不複雜，所以我們想先試驗看看用一般的CNN的模型進行訓練
- 因為要讓CNN的模型進行訓練，我們將原本收集到的資料在多做一個處理
  - 原本的資料（32*32）的圖片兩張
  - 兩張合併再一起變成（64*32）
- 因為windows size是12 所以我們一次把12個frame產出來的圖疊在一起，並把12個frame當成channel size並輸入至Conv2D

Model Version 1(2/3)

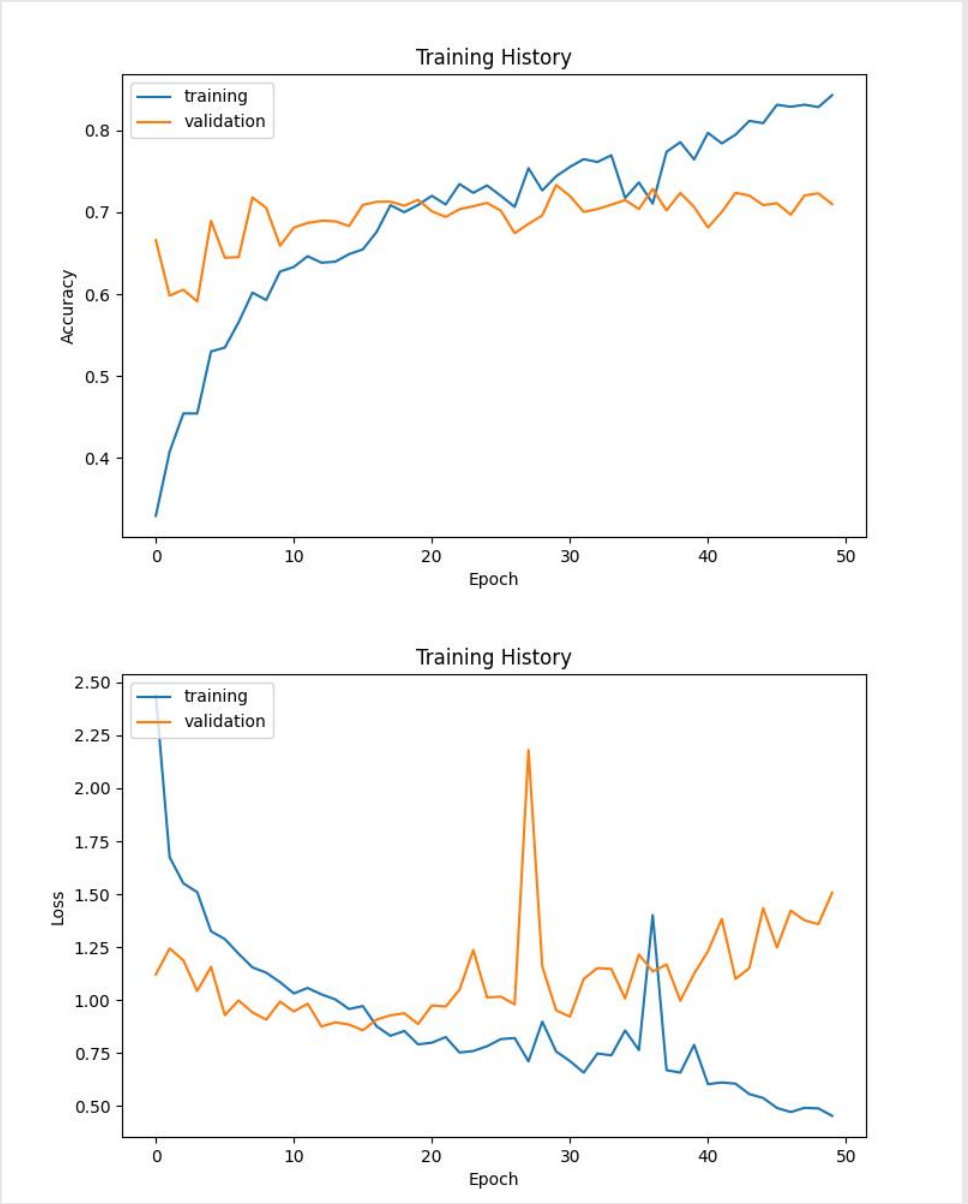
模型架構圖如下：



Model Version 1(3/3)

訓練到最後在validation set上得到
Accuracy：0.7097
Loss：1.507

可以觀察到模型最後有overfitting的情況，所以我們下一版模型進行調整

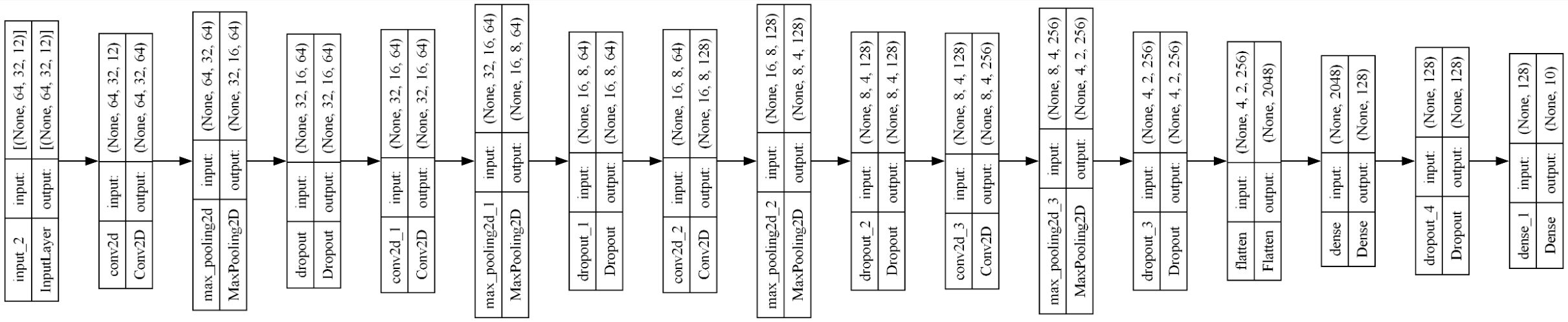


Model Version 2(1/3)

- 根據我們Version 1的架構，我們觀察到模型後面有明顯的overfitting的情況，所以我們嘗試解決這個問題
- 解決overfitting的一個好用的方法是dropout，也就是每次訓練的時候隨機關掉幾個neuron，讓模型不會過度fitting在training data上
- 所以我們在原本的模型中，每個pool layer後面加上一個0.3的dropout layer，並在兩層fully connected layer中間加上一個0.5的dropout layer

Model Version 2(2/3)

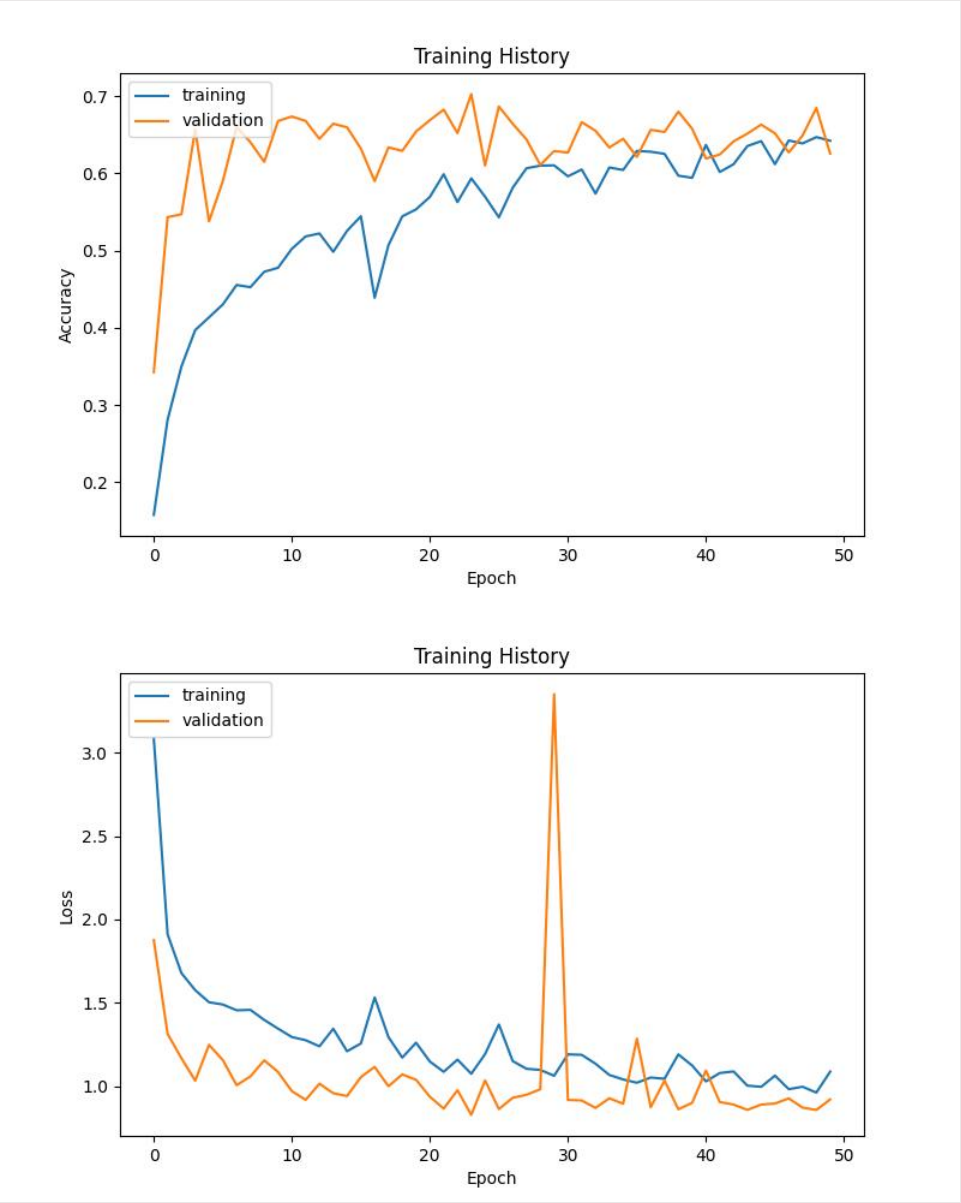
模型架構圖如下：



Model Version 2(3/3)

訓練到最後在validation set上得到
Accuracy：0.6255
Loss：0.9214

可以發現模型overfitting的情況雖然有一定程度的被抑制，但是模型效能卻降低了，我們覺得CNN可能極限就在這邊，所以我們決定換一個模型架構試驗看看

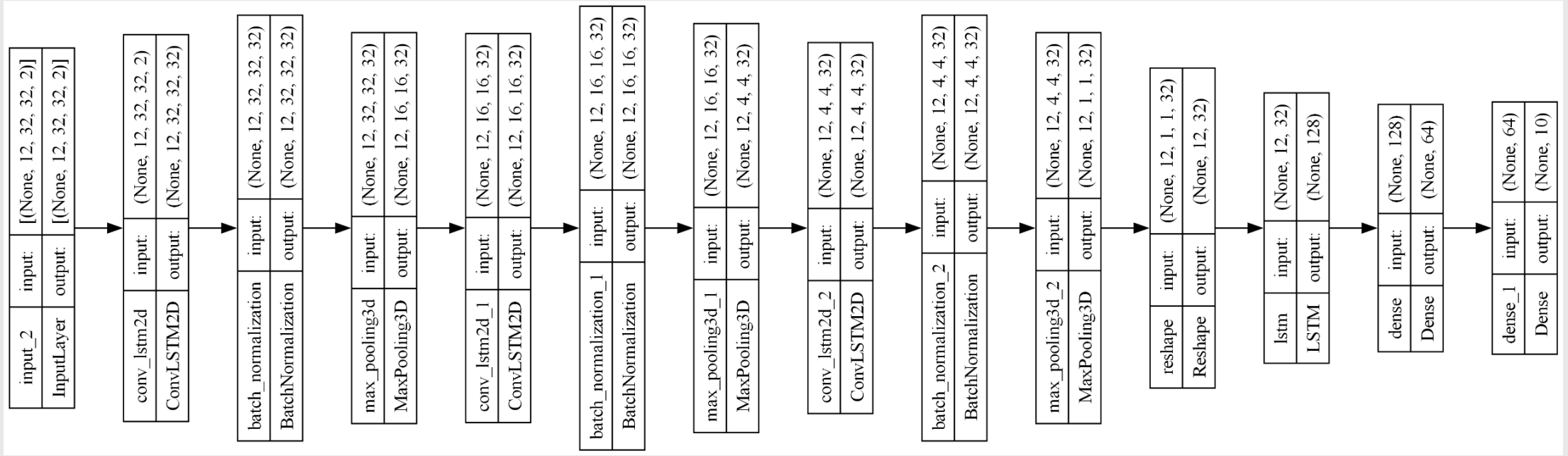


Model Version 3(1/3)

- 在我們試驗完CNN的模型後，我們開始思考到底為什麼感覺模型已經到極限了，模型的accuracy表現還是不佳，我的得到一個結論，原先單純的CNN應該是不行的，因為影像應該是一種具有time step的概念的資料
- 所以應該要加上可以處裡time step的模型上去，經過一番查找後找到ConvLSTM[1]這個模型可以使用
- 所以我們簡單搭建了一個ConvLSTM試驗看看效果，看有沒有辦法達到更好的效能

Model Version 3(2/3)

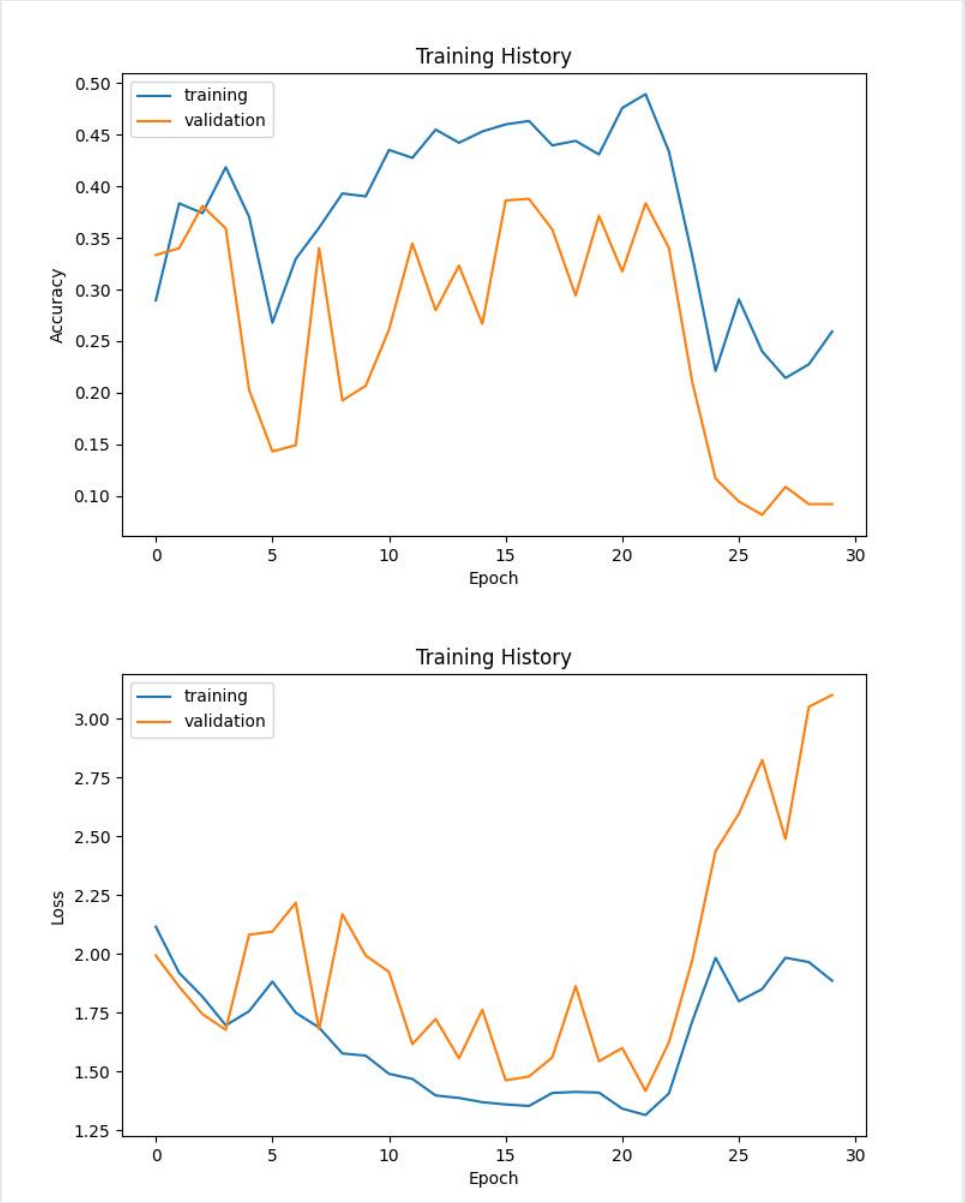
模型架構圖如下：



Model Version 3(3/3)

訓練到最後在validation set上得到
Accuracy：0.0922
Loss：3.1

我們發現，模型訓練到一半發生梯度爆炸，accuracy就跌下去了，loss也突然就上升了，所以我們在想，會不會是模型本身效能，或者是資料沒有打的足夠亂，以及optimizer的問題

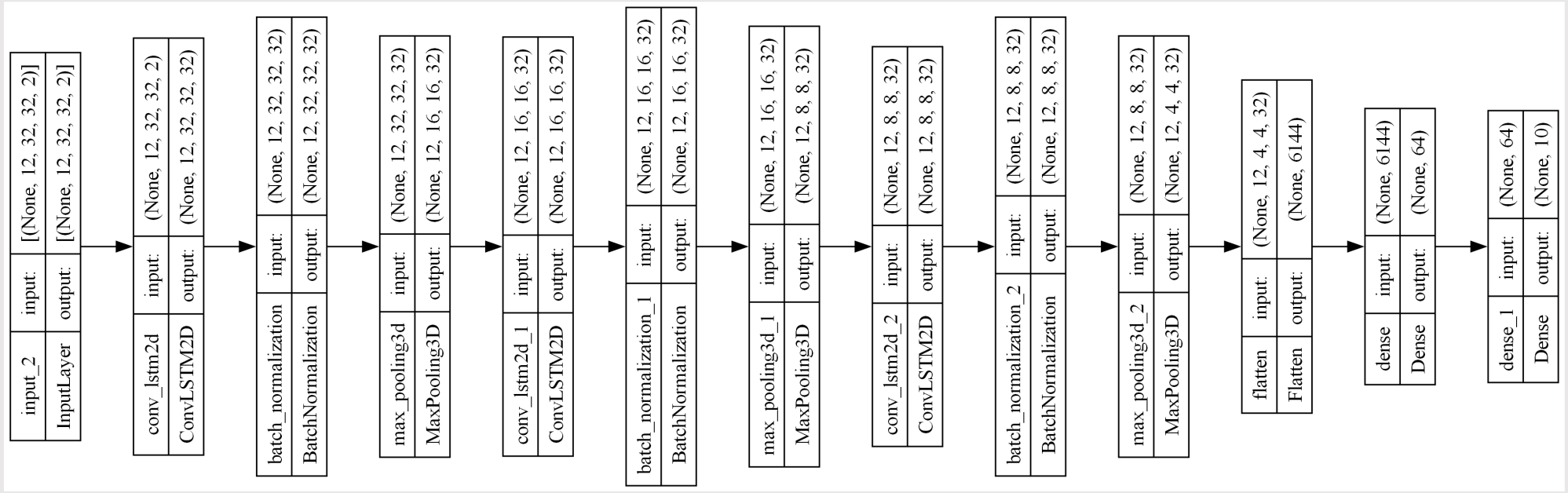


Model Version 4(1/3)

- 在我們Version 3的模型版本中，我們發現了模型訓練到中途會發生梯度爆炸；此外，模型的表現也不盡理想，最高的accuracy只有0.5左右，所以我們決定調整模型的hyperparameter來試驗看看，能不能改善這個問題
- 我們想要先試著試驗看看，是不是因為模型不夠複雜所以導致表現不佳或者是因為其中我們加了LSTM layer所導致
- 所以我們把原先模型的kernel size拉大，增加模型參數量，並去掉LSTM layer，來看看這樣對模型的表現有什麼樣的影響

Model Version 4(2/3)

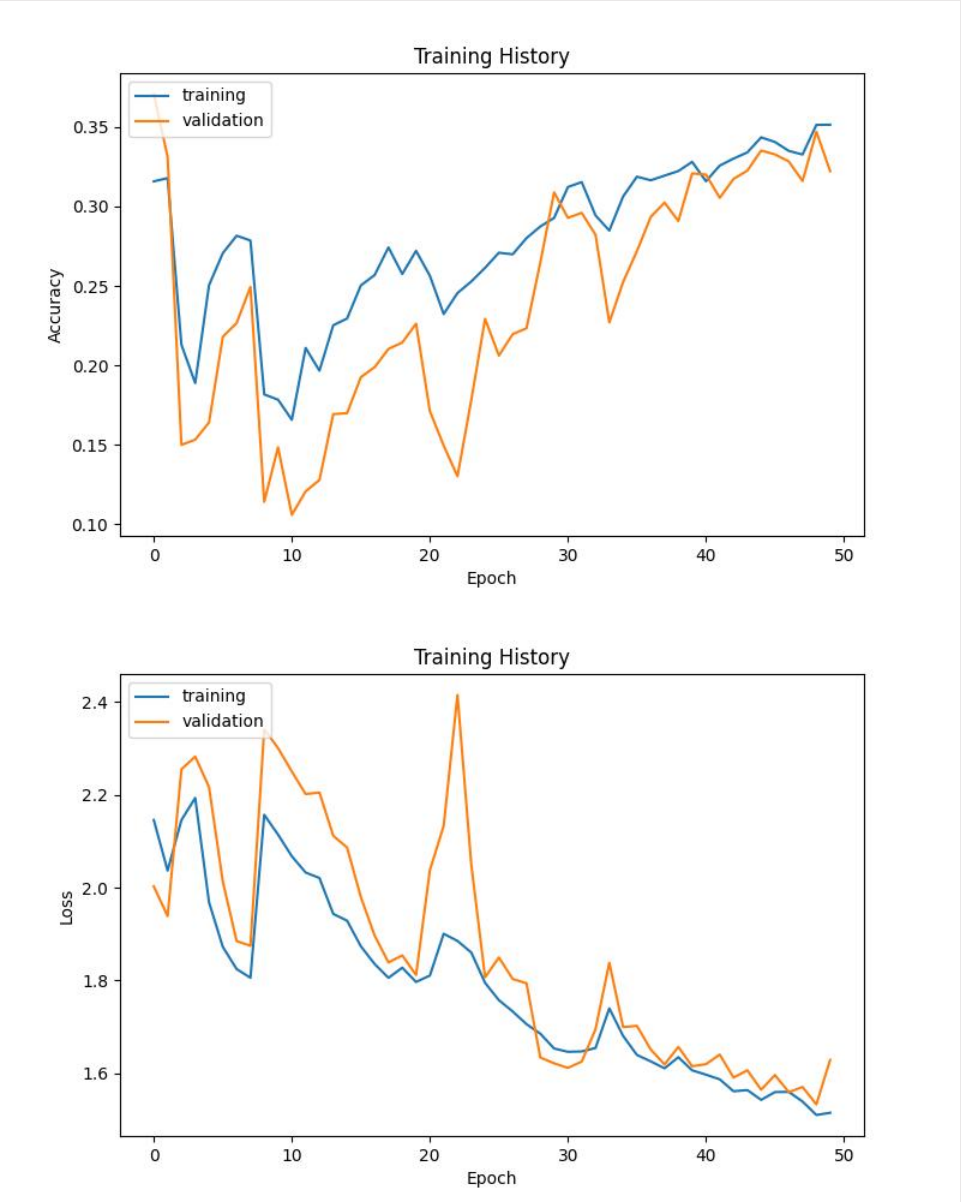
模型架構圖如下：



Model Version 4(3/3)

訓練到最後在validation set上得到
Accuracy：0.3221
Loss：1.6286

這次明顯有改善再Version 3的時候發生的問題，但是模型的效能仍舊十分不理想，所以在基於這個模型進行後面資料打亂跟其他optimizer的測試

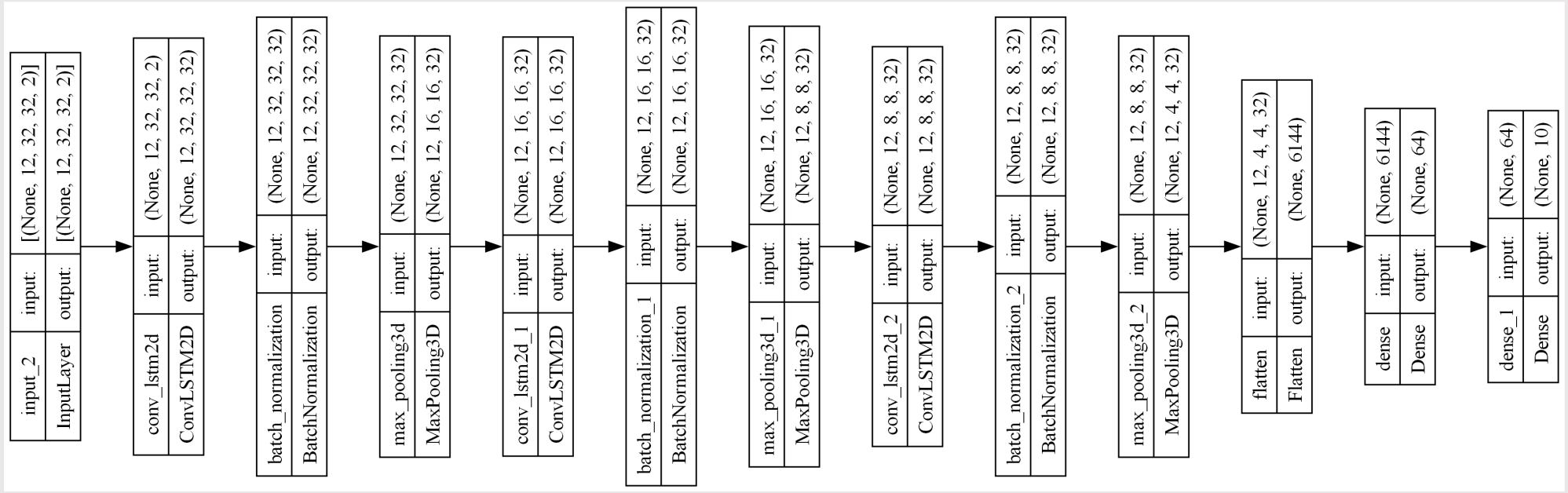


Model Version 5(1/3)

- 在Version 3的模型中，我們發現模型訓練到一半會發生accurac以及loss暴跌的情況，我們在Version 4中透過去除LSTM layer且增加模型的強度解決了這個問題，但是模型的效果人仍舊十分差強人意
- 我們猜測，會不會是我們原本打亂的方式有問題，所以想使用Tensorflow原生的套件重新將資料打亂，看效果會不會有明顯的變化
- 為了達到這個效果，我們將原本的資料全部按照label排序好，並在在打開Tensorflow的shuffle，達到重新打亂的效果

Model Version 5(2/3)

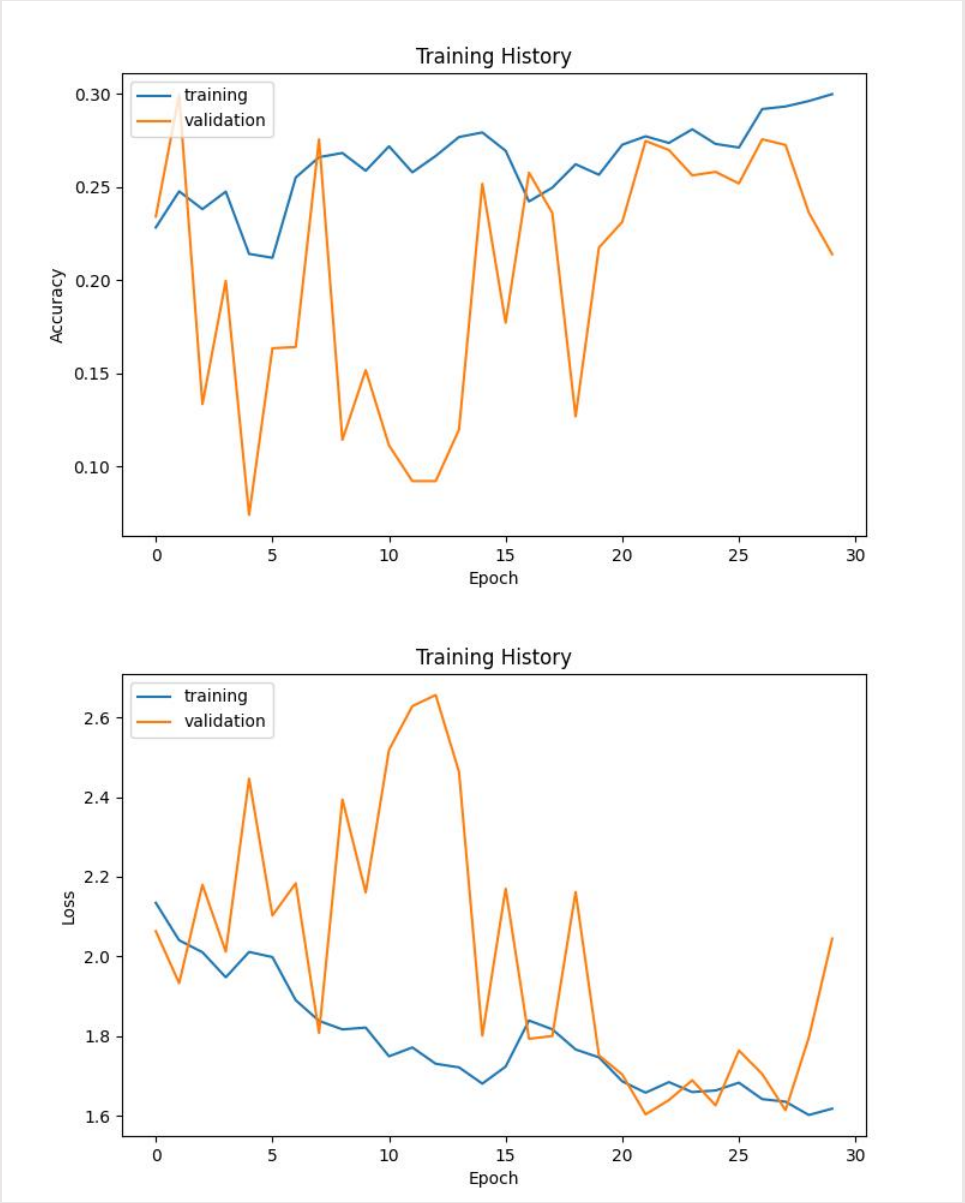
模型架構圖如下：



Model Version 5(3/3)

訓練到最後在validation set上得到
Accuracy：0.214
Loss：2.0447

我們發現，重新透過Tensorflow內建的套件來打亂，模型訓練的效果並沒有比較好，所以我們後續就不去重新再打亂資料

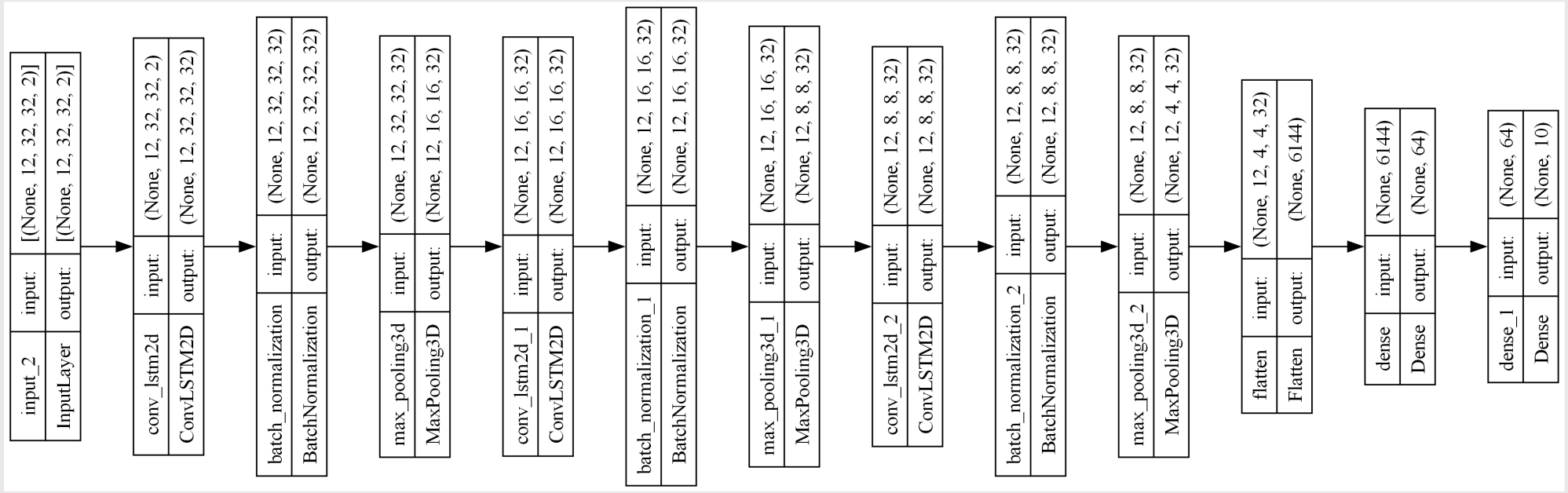


Model Version 6(1/3)

- 剩下一個測試，就是基於Version 4的模型架構去換optimizer，看看訓練效果上會不會有更好
- 我們前面所有測試都是使用最常見的optimizer：Adam，在更換上我們根據經驗，我們選用Adamax這個optimzier進行訓練

Model Version 6(2/3)

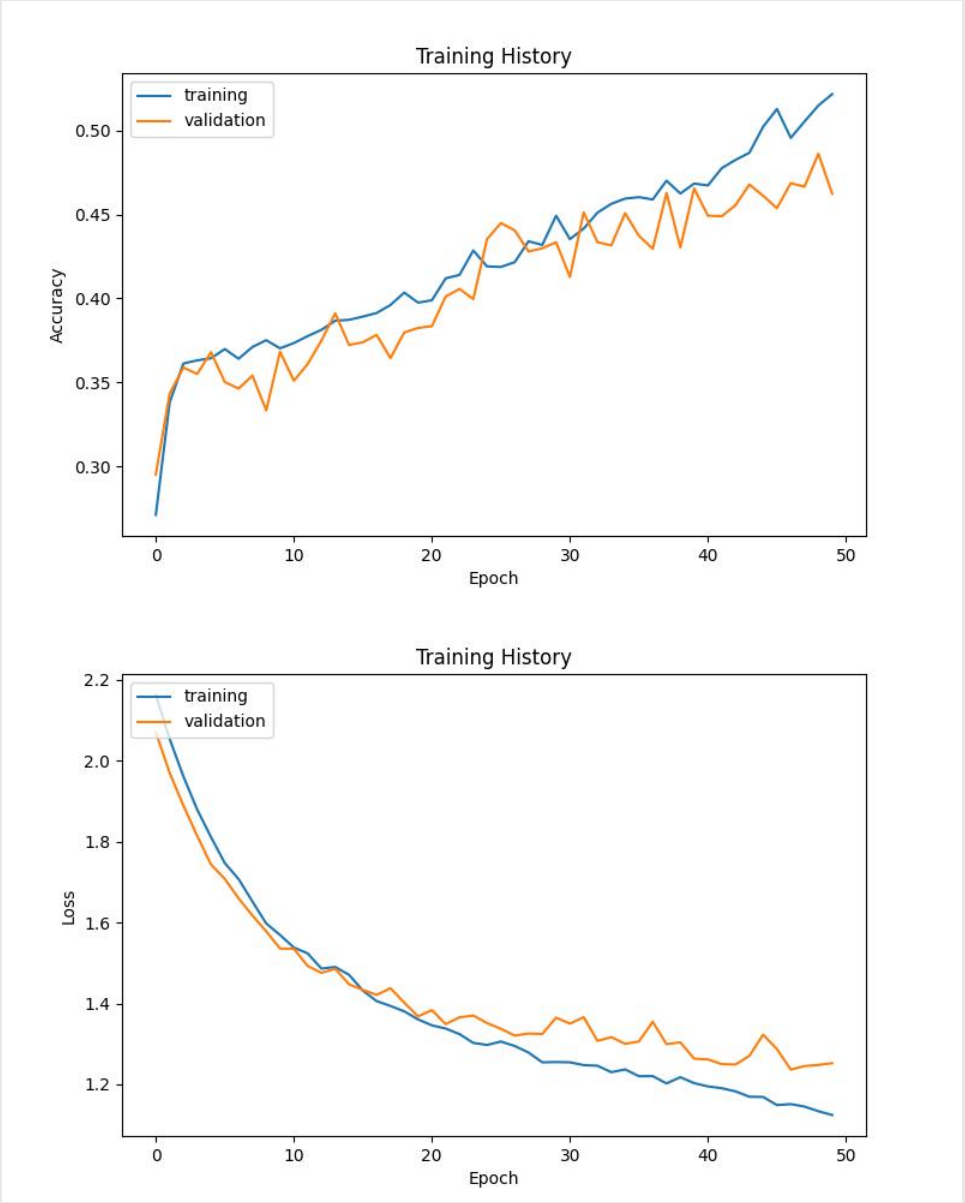
模型架構圖如下：



Model Version 6(3/3)

訓練到最後在validation set上得到
Accuracy：0.4623
Loss：1.2524

我們發現，其實learning curve有更平整，但是最後的accuracy並沒有十分令人滿意，所以我們決定放棄這個模型，改用其他方式重新架構我們的模型

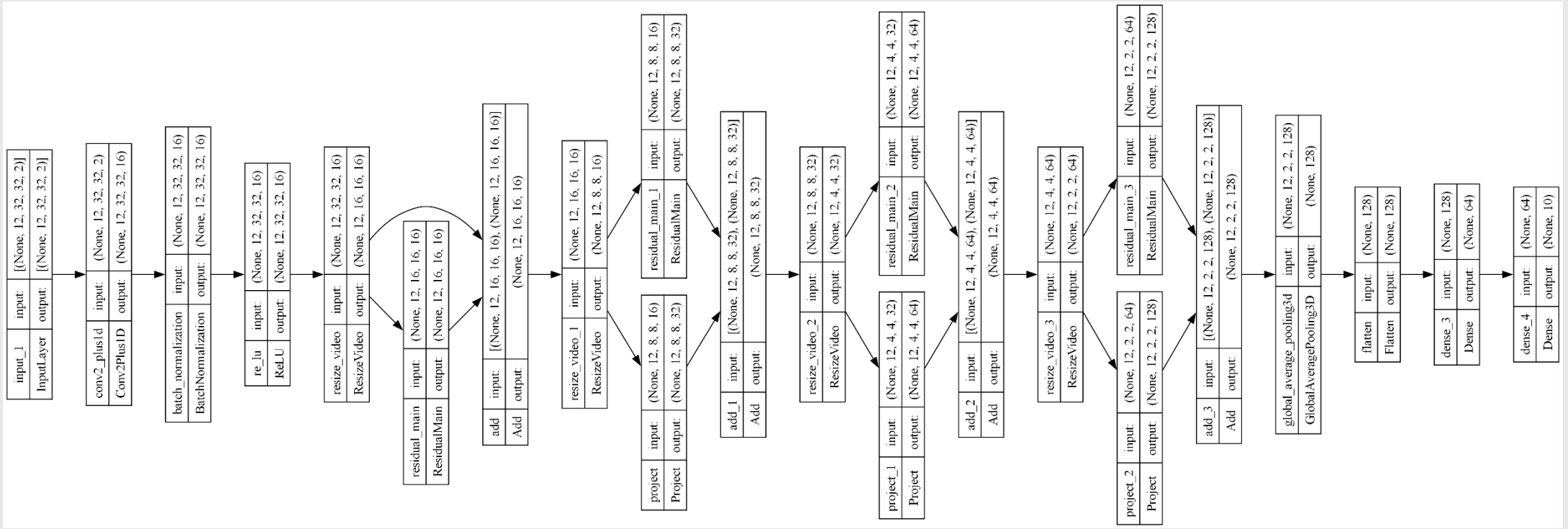


Model Version 7(1/3)

- 我們開始重新尋找各式各樣不同的模型，後來發現一個有趣的模型，他是從原本的CNN出發，但是把原本的3維的影片概念，用類似Depthwise Separable Convolution的概念，並同時融合ResNet的概念，將原本Conv3D同時對Spatio部分跟Temporal部分進行掃描，分開來運算[2]
- 基於這樣的一個想法，我們重新架構了我們的模型，並得到Version 7的模型進行測試

Model Version 7(2/3)

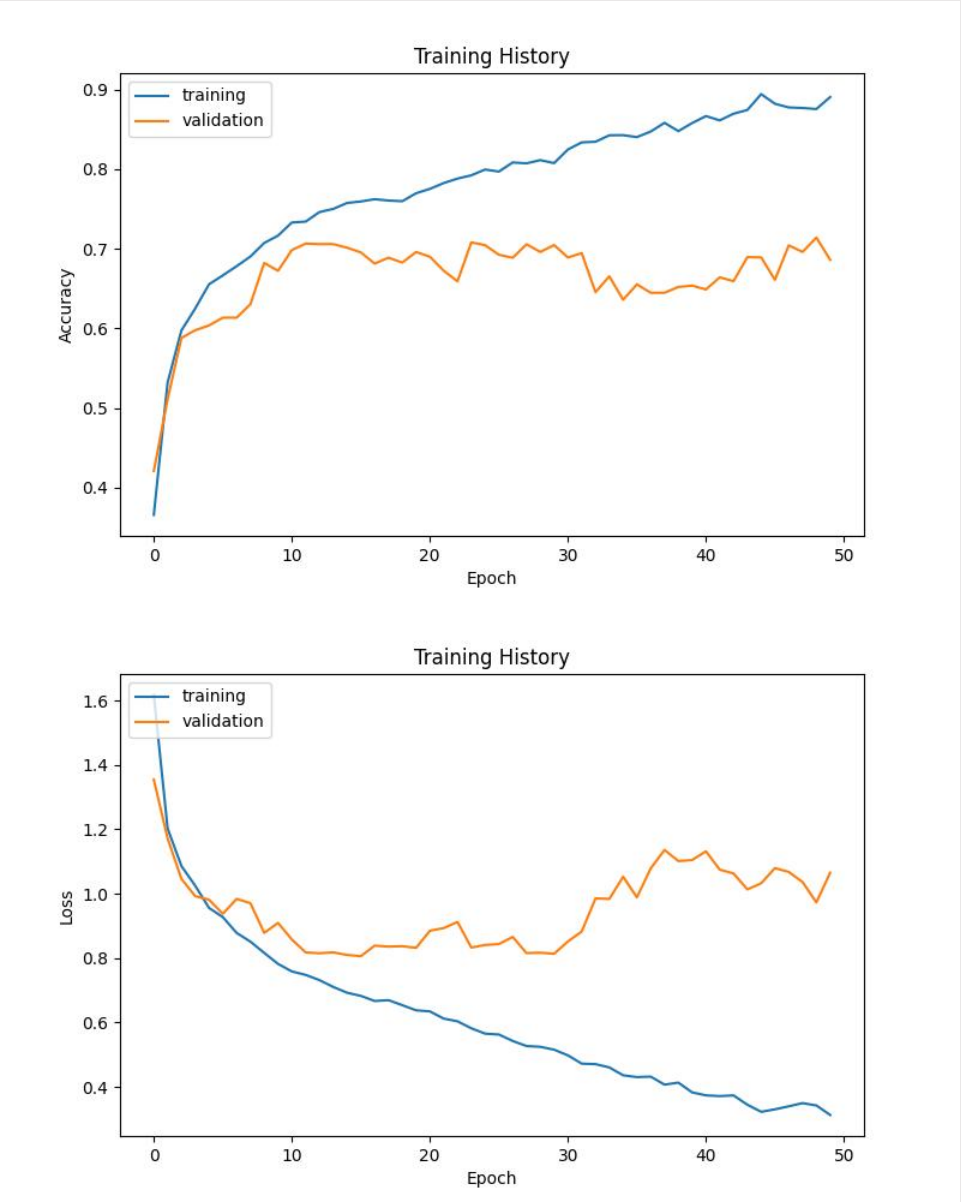
模型架構圖如下：



Model Version 7(3/3)

訓練到最後在validation set上得到
Accuracy：0.6861
Loss：1.065

從learning curve我們發現，模型訓練到最後面有明顯overfitting的情況，但是單看training set上的表現十分的不錯，所以我們決定來嘗試解決這個模型overfitting的問題

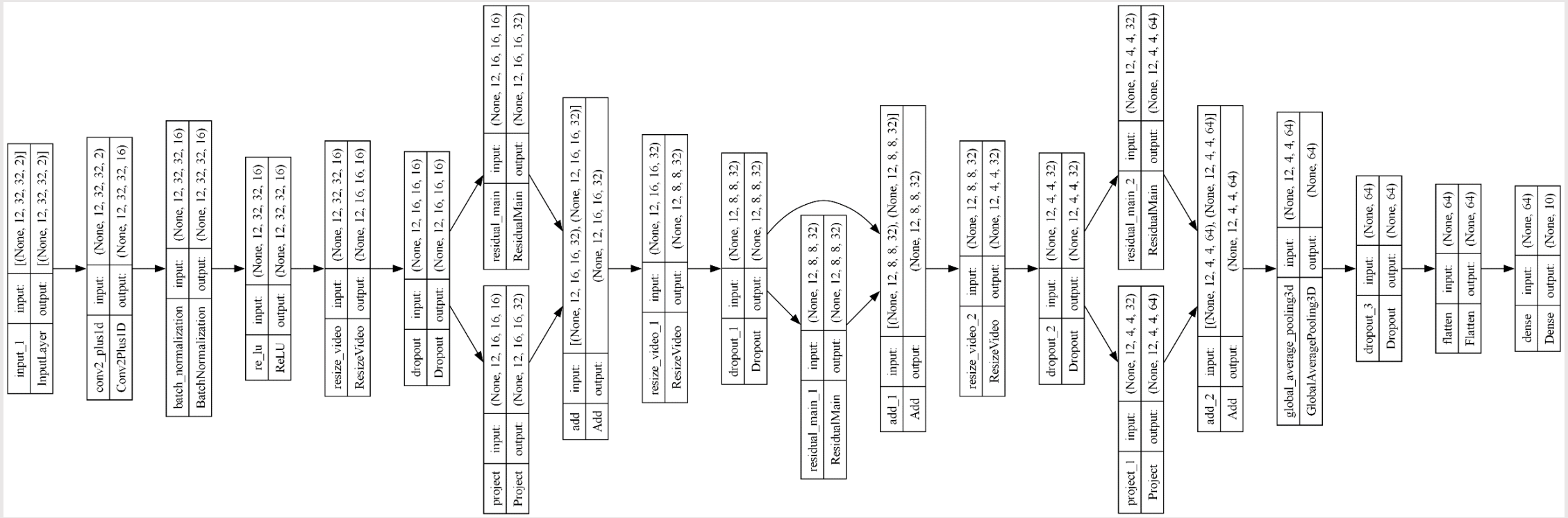


Model Version 8(1/3)

- 在Version 7的模型中，可以看到模型有明顯的overfitting的情況，我們希望可以解決這個問題
- 正如我們在CNN的模型中解決overfitting的時候，我們選用的方法，就是適度的在模型中加入dropout並減少模型參數量來嘗試解決overfitting的問題
- 我們在模型中部分地方加入了0.3的dropout layer來試驗看看，能不能有效的降低模型overfitting的問題

Model Version 8(2/3)

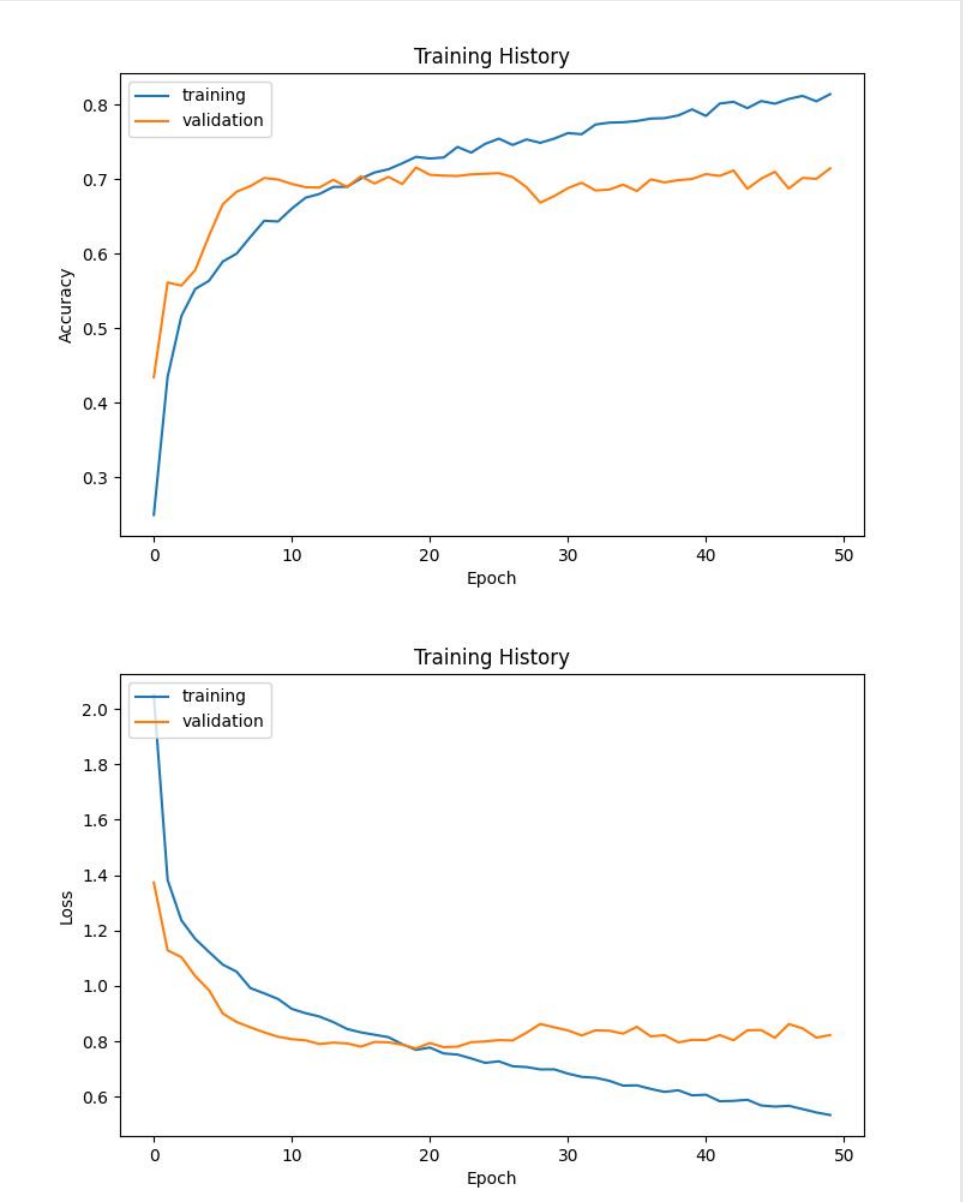
模型架構圖如下：



Model Version 8(3/3)

訓練到最後在validation set上得到
Accuracy：0.7143
Loss：0.8224

從這次的learning curve我們可以發現，雖然還是有一定程度的overfit-ting的情況，但是相較於Version 7而言，已經有明顯的改善的情況了

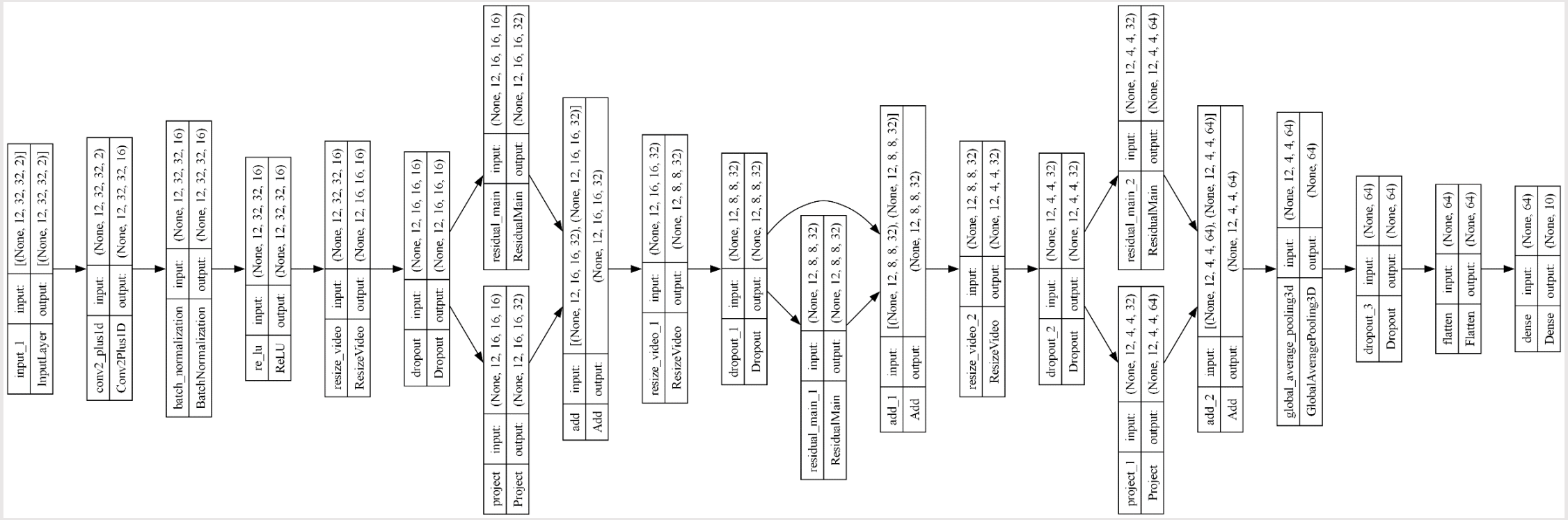


Model Version 9(1/3)

- 在Version 8的模型，其表現實已經相當不錯了，而且參數量相較於前幾個模型，也減少很多
- 但是考慮到實際demo時模型要不停讀取資料並做預測，我們擔心就這樣的參數量可能無法負荷，於是我們想實驗看看，如果繼續把參數量降低，在提高模型的預測速度度同時，其效能表現會如何

Model Version 9(2/3)

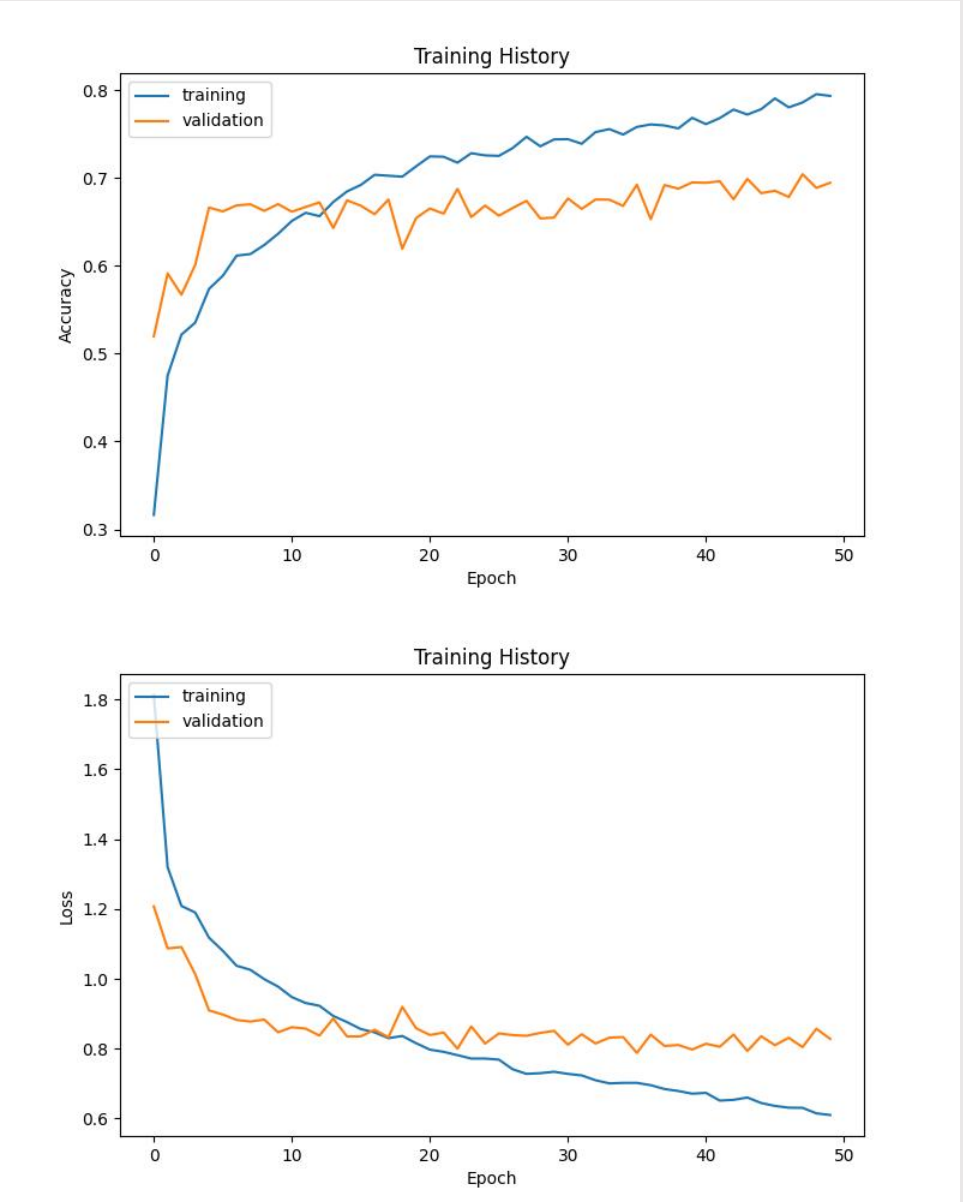
模型架構圖如下：



Model Version 9(3/3)

訓練到最後在validation set上得到
Accuracy：0.7265
Loss：0.7521

我們在learning curve上發現，雖然還是有觀察到overfitting的情況，但是相較於Version 8而言，並沒有明顯加劇的情況，而且效能表現也差不多，甚至更好，礙於時間因素所以R(2+1)D的CNN模型就測試到這邊

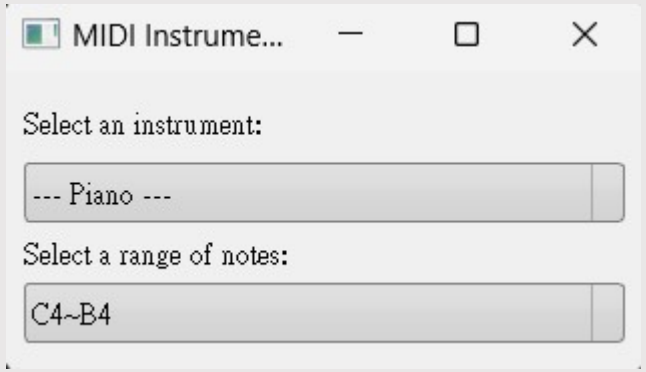


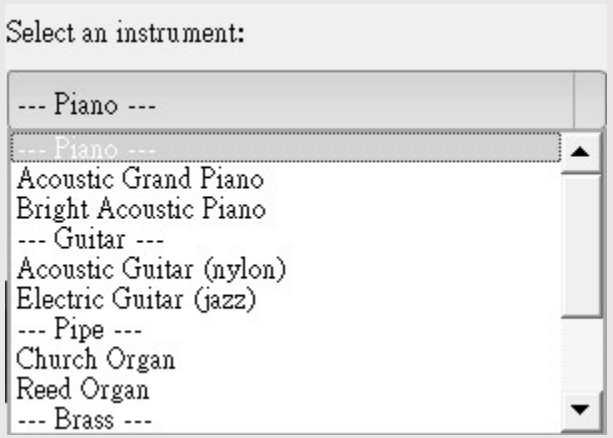
Final Model

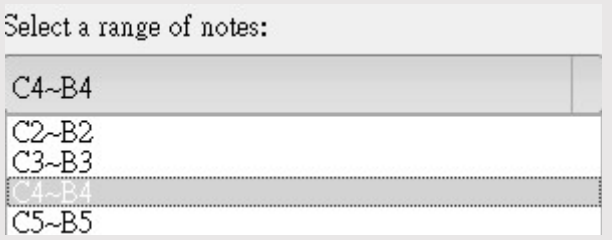
- 我們合計有九種不同版本的模型，我們可以發現Version 1以及Version 9的模型的效果都十分不錯，兩者甚至相差無幾
- 但是在考量參數量後，我們會發現模型Version 1的參數量遠大於Version 9的模型，所以我們最終選定Version 9作為我們最終的模型，其他模型就不放入我們後續的系統中使用

Music GUI

- 原先在設計上，我們只想說讓模型預測出的東西，只能撥放鋼琴的聲音，但後來發現使用PyGame去調用Windows的合成器，可以調用十分多種不同的聲音，所以我們決定多加其他樂器以及不同聲部
- Python本身撰寫圖形化介面就十分容易，所以簡單撰寫了一個可以讓使用者簡單變化演奏的樂器以及音部的圖形化介面







Difficultly(1/2)

- 因為我們電腦的記憶體以及算力有限，我們發現當一筆資料的總frame一旦太高，電腦處理會非常花時間，但是一次一個影片包含多種label比較符合實際使用情況，所以變成資料在收集上的難度變得非常高（平均一個動作/換動作只有不到一秒的反應時間）
- 且由於一個影片的總frame數有限，所以導致在轉換動作的時候容易會發生操作失誤，我們推測也因此，導致模型訓練的效能表現沒法太高

Difficultly(2/2)

- 我們都在各地遠端線上工作，在串接所有的code再一起的時候，我們發現一個情況，官方的sample code經過壓縮檔傳送後，在另外一台電腦解壓縮，會發生.dll被列為不信任的檔案，之後就無法執行了
- 演奏樂器本身是一個十分精細的動作，但是我們發現毫米波雷達的偵測器有點難以收集到很多精細動作的變化

Conclusion

Conclusion

- 這次在辨識手勢的模型實際部屬後的效果十分差強人意，我們推測可能我們在資料收集有一些問題，可能資料的數量需要再增加，並應該分開label的方式進行收集，使動作可以更加完整以及精細，也不會讓電腦在處裡上需要花太多時間。雖然這次實作出來的效果並非十分優越，但是仍舊有一定程度的可以運作，撇除模型效能不佳外，其他部分都有正常的運作，但仍舊對於毫米波的未來提出了一個可能發展的趨勢，以及對於未來有想要做類似物品的人提出了一些可能。

Reference

Reference

- [1] Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W., & Woo, W. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. Neural Information Processing Systems.
- [2] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. (2017). A Closer Look at Spatiotemporal Convolutions for Action Recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6450-6459.
- [3] Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lucic, M., & Schmid, C. (2021). ViViT: A Video Vision Transformer. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 6816-6826.

Appendix

Appendix A

- Source Code：
  https://github.com/ase12345636/2024-mWave-AI-Otamatone