基於機器學習進行網路切片的辨識

(Recognition of network slices based on machine learning)

作者：張毅偉、曾宇晨

系級：通訊碩二

學號： M0991570、M1005783

開課老師：陳立勝

課程名稱：機器學習於物聯網之設計與應用

開課系所：通訊工程學系

開課學年：111 學年度 第 1 學期

中文摘要

隨著無線通訊的蓬勃發展，目前擁有大頻寬、高傳輸、高擴充性的 5G 時代及未來的 6G 時代，使得人們在使用無線通訊網路的依賴性與日俱增，相較於 4G-LTE 不僅提升網路速度在頻寬上也更大，當然僅只有這些演進無法彰顯 5G-NR 的強大。5G-NR 雖然在網路速度、頻寬、設備的連結性數量相較 4G-LTE 有大幅的提升但是網路資源依然是有限的，所以在 5G 網路裡有了網路切片的產生，何謂網路切片呢?網路切片主要的功能是將無線網路的資源透過虛擬化的方式將其功能進行分割，而目前根據 3GPP 在 TS 28.541 中提出了 5 項切片服務為 URLLC(Ultra-Reliable and Low-Latency Communications)、eMBB(Enhanced Mobile Broadband)、MIoT(Massive Internet of Things)、V2X (Vehicle-to-Everything)、HMTC(High Performance Machine-Type Communications)。

本次專題所使用的切片類別有 URLLC、eMBB、MIoT 將所需要的無線網路服務進行這三種分類，當輸入層收到了 16 項參數會進行網路切片類型的選擇，再透過 tensorflow 建立一個 ANN(Artificial Neural Network)的模型，模型訓練完後模型的輸出層會選擇出應用適合哪項網路切片的類型，最後並進行資料驗證及比對來做正確率的分析。

關鍵字：URLLC(Ultra-Reliable and Low-Latency Communications)、MIoT(Massive Internet of Things)、eMBB(Enhanced Mobile Broadband)、tensorflow、ANN(Artificial Neural Network)

Abstract

With the booming development of wireless communications, the current 5G era with large bandwidth, high transmission, and high scalability and the future 6G era have made people more and more dependent on the use of wireless communication networks, which not only improve the network speed and bandwidth compared to 4G-LTE, but of course, only these evolutions cannot show the power of 5G-NR. Although 5G-NR has significantly improved the network speed, bandwidth, and number of connected devices compared to 4G-LTE, the network resources are still limited, so network slicing is created in 5G networks. What is network slicing? The main function of network slicing is to partition the resources of wireless network by virtualization, and according to 3GPP, five slicing services are proposed in TS 28.541: URLLC (Ultra-Reliable and Low-Latency Communications)
According to 3GPP in TS 28.541, five slicing services are proposed: URLLC (Ultra-Reliable and Low-Latency Communications), eMBB (Enhanced Mobile Broadband), MIoT (Massive Internet of Things), V2X (Vehicle-to-Everything), and HMTC (High Performance Machine-Type Communications).
The slicing categories used in this project are URLLC, eMBB, and MIoT. These three categories are used to classify the required wireless network services. After the model is trained, the output layer of the model will select which type of network slice is suitable for the application, and finally, the data will be verified and compared to do the correctness analysis.

Keyword：URLLC(Ultra-Reliable and Low-Latency Communications)、MIoT(Massive Internet of Things)、eMBB(Enhanced Mobile Broadband)、tensorflow、ANN(Artificial Neural Network)

目　次

第一章 研究動機與目的

5G NR 演進不僅改變了人類的生活習慣，各類的網路設備及服務、數據的流量需求也日漸倍增。在 4G LTE 的基礎上 5G NR 技術不僅具有更大的頻寬、更高的網路流量、可與更多的設備進行連接，但是當每個用戶或設備都使用這三種網路具有的功能，這樣網路資源還是會被消耗殆盡的，因此 5G NR 在提升網路的功能外，也發展出了新的功能而這個功能即為網路切片，網路切片就是將有限的頻譜資源透過虛擬化的方式進行有效的分配，透過這些分配可以將網路的功能分配給適合的應用或設備。

而本專題主要透過機器學習來進行網路切片所適用的應用進行分配，透過 Tensorflow 實作神經網路模型來進行「網路切片類別辨識」，而這次採用的資料集的出處在 kaggle 這個資料及網站，此網站有許多適用機器學習或深度學習模型的訓練的資料集，透過 Network Slicing Recognition 這個資料集來訓練我們撰寫的模型架構，透過訓練完的權重即使出現了識別切片服務功能故障，也能透過連接的應用自動檢測最合適的網路切片。

第二章 先前方法研討

2.1 資料集作者的作法:

本專題使用的資料集來自 Kaggle 網站，此網站是一個數據建模和數據分析的競賽平台，會有許多不同的研究學者或企業於此發布多種類別的資料集。其中我們所使用的資料集，其作者使用 Microsoft 所開發的 Fast and lightweight AutoML (FLAML)來對此資料集進行機器學習的訓練。

關於安裝及使用 FLAML 步驟如下:

1. 安裝方式:

關於 FLAML 的 python 最低版本需要使用 python3.7，並打該 CMD 透過 python 中安裝的套件 pip 來進行安裝，安裝指令如下。

需 python3.7 以上	**pip install flaml**

2. 使用方法 [1]:

從 flam 的資料庫中 import AutoML 的函式，在訓練的不分透過 fit()這個函式，函式如下圖中的參數分別為訓練輸入層、訓練輸出層、還有任務這裡選用的是分類最後一個參數為訓練次數，而這個資料庫的作者訓練的 900 次。

```
from flaml import AutoML
_automl = AutoML()
_automl.fit(x_train,y_train,task = "classification",time_budget=900)
```

2.2 網路切片說明:

在 5G 網路中 UE 透過 NSSAI(Network Slice Selection Assistance Information)中的 S-NSSAI(Single Network Slice Selection Assistance Information)透過 N2 介面向 5G 核心網路中的 AMF(Access and Mobility Management Function)來進行網路切片的類型，這個 Function 主要是在管理及控制 UE 的接入核心網路的功能，而目前在 3GPP 中 S-NSSAI 代表的數值包含了 eMBB、URLLC、MIoT、V2X、HMTC 透過表 1 及表 2 [2]可以看到在 NSSAI 中包含了許多的 S-NSSAI 而在 SNSSAI 中有 8 位元的 SST 及 24 位元的 SD，SST 代表切片類別 SD 則是將類似的 SST 進行更細微的劃分。

NSSAI IEI	octet 1
Length of NSSAI contents	octet 2
S-NSSAI value 1	octet 3 octet m
S-NSSAI value 2	octet m + 1 octet n
⋮	octet n + 1 octet u
S-NSSAI value n	octet u + 1 octet v

表 1. NSSAI 中的元素

SST	**SD**
8 bits	24 bits

S-NSSAI

表 2. S-NSSAI 中的元素

2.3 gNB 對 AMF 的選擇:

如下(圖 1 [3]我們可以看到 gNB 透過 N2 介面可以選擇有支援該 slice 的 AMF，當 UE 透過 gNB 選擇 2、3 的 slice 則他會找到有同時支援 2、3、4 的 AMF。

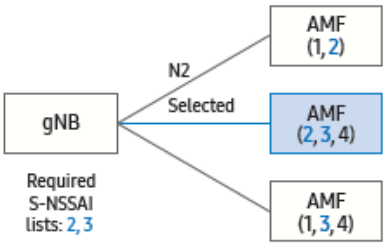


圖 1. gNB 透過 N2 介面選擇適合 UE 的 AMF

第三章 問題定義

3.1 關於 dataset:

透過網路切片的說明可以了解到關於網路切片相關過程及內容，而透過這個內容和這次要實作的資料集我們定義了關於網路切片和 5G 相關應用的關係，透過這個關係我們可以將這個資料集的的切片類別作為模型的輸出，而 LTE/NR、Time、Packet Loss Rate、Packet delay 等作為這次訓練模型的輸入詳細的內容如表 3 [4]

LTE/5g Category : LTE/5g 類別	AR/VR/Gaming : AR/VR/遊戲 (Yes : 1 No: 0)
Time : 時間 (幾點 12:00p.m.為 0 依此類推，依 24 小時制 ex : 晚上 9:00 為 數字 21)	Healthcare : 健康照護 (Yes : 1 No: 0)
Packet Loss Rate : 封包遺失率(未收到封包總數/發送封包總數)	Industry 4.0 : 工業 4.0 (Yes : 1 No: 0)
Packet delay : 封包延遲(ms) (收到封包的時間)	IoT Devices : 物聯網裝置 (Yes : 1 No: 0)
IoT : 是否為物聯網 (Yes : 1 No: 0)	Public Safety : 公共安全 (Yes : 1 No: 0)
LTE/5G : LTE 或 5G (LTE : 0 5G : 1)	Smart City & Home : 智慧城市或家庭 (Yes : 1 No: 0)
GBR : 有保證 bit Rate (Yes : 1 No: 0)	Smart Transportation : 智慧交通 (Yes : 1 No: 0)
Non-GBR :沒有保證 bit Rate (Yes : 1 No: 0)	Smartphone : 智慧型手機 (Yes : 1 No: 0)
slice Type : 切片類別 (eMBB : 1 mMTC: 2 URLLC : 3)	

表 3.資料集說明

根據上述資料集內容及對於切片選擇的要求，我們可以將訓練集分為 16 個 X train 跟 1 個 Y_train。

第四章 規劃、設計之架構與方法

4.1 基礎架構:

透過 Google 所開發的 Tensorflow 套件來建立一個 Sequential Modle，為甚麼是使用 Sequential Modle 呢?主要是因為他是單輸入單輸出的模型架構(如圖 2，但如果是要做多輸入多輸出的模型架構就需要透過 Functional API 來建立模型，這次的輸入層有 14 個參數透過 reshape 的方式將參數加入到輸入層，而輸出層則為 3 個參數一樣也透過 reshape 的方式放入模型的輸出層。

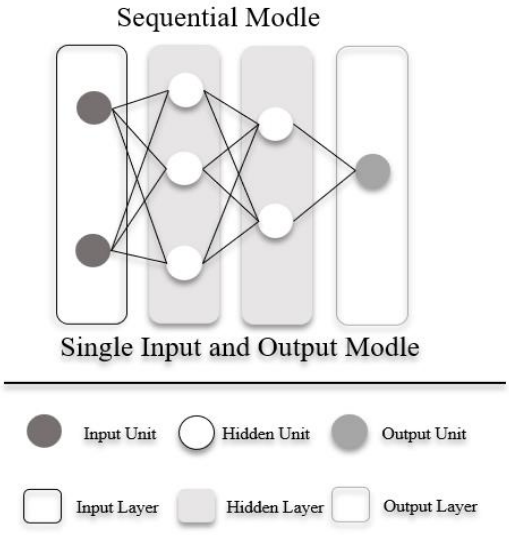


圖 2. Multi Input and Output Modle

4.2 訓練流程:

透過下方的訓練流程 圖 3，將 data 讀入 inputX 和 inputY 並使用亂數索引直將索引直打亂，再將資料集 70%分為 Training data 和 20% 的 Validation data 10%的 Testing data 之後將訓練集放入 Sequential Modle 內，此 Modle 共有 14 個參數輸入層和 2 個隱藏層集 3 個參數輸入層，在訓練的過程將損失函數透過優化器優化模型的訓練在最後在對 Validation data 進行比對來確定準確率，最後訓練出來的權重透過 Testing data 進行驗證並繪出指標函數圖。

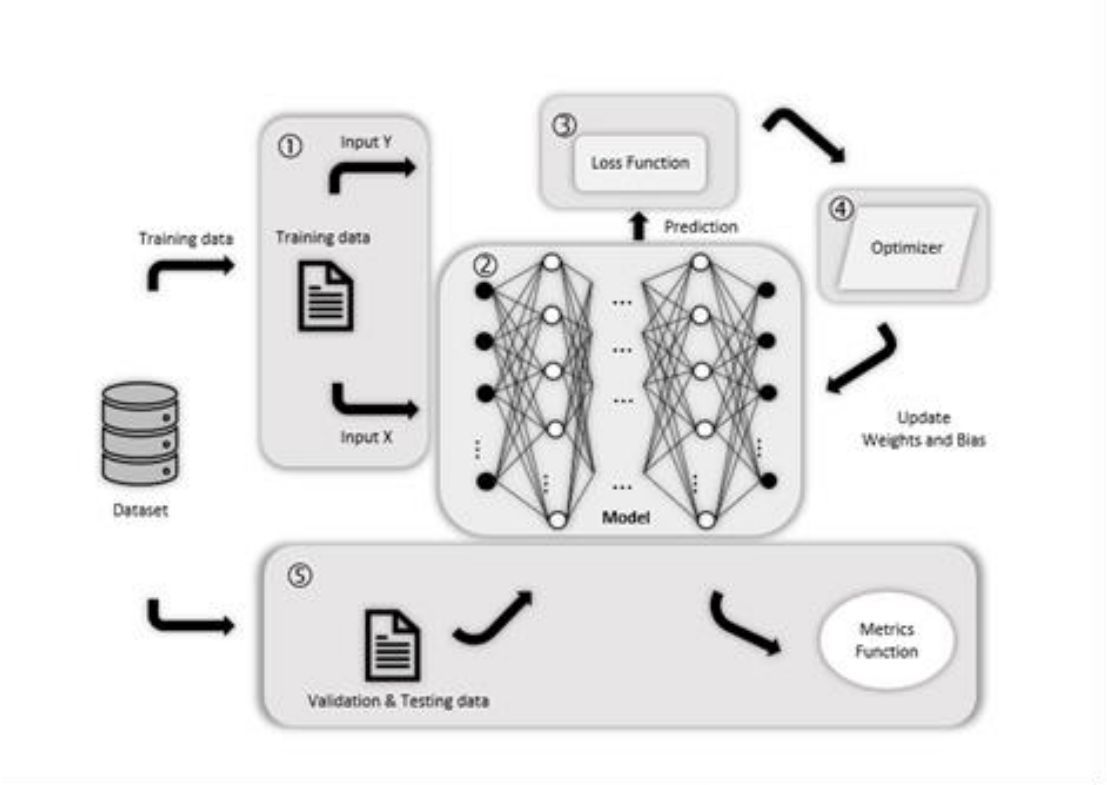


圖 3. 模型訓練流程

4.3 模型建立:

本專題對於 Modle 的建立(圖 4 共有 1 個輸入層和 1 個隱藏層集 1 個輸入層，而 1 到 3 層皆為為全連結層，第一層隱藏層有 64 個神經元，而第二層有 32 個神經元，第三層有 3 個神經元，而激活函數的選擇 1 二層皆為 relu 而最後一層因為分類問題所以激活函數為 softmax。

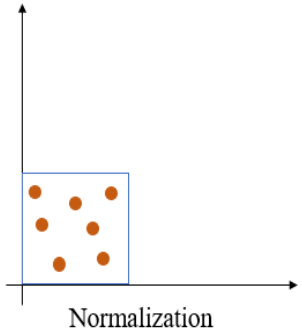
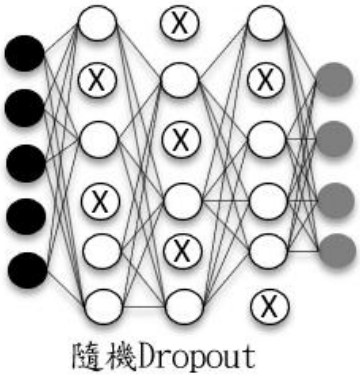
```
In [83]: # 建立一個Sequential型態的model
model3 = keras.Sequential(name='model3')
model3.add(layers.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001), kernel_initializer = 'uniform',
                        activation='relu', input_shape=(14,)))
model3.add(layers.Dropout(0.385))
model3.add(layers.Dense(32, kernel_regularizer=keras.regularizers.l1(0.001), kernel_initializer = 'uniform', activation='relu'))
model3.add(layers.Dropout(0.364))
model3.add(layers.Dense(3, kernel_initializer = 'uniform', activation='softmax'))
# 顯示網路模型架構
model3.summary()

Model: "model3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 64)                960
_________________________________________________________________
dropout_24 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_37 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_25 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_38 (Dense)             (None, 3)                 99
=================================================================
Total params: 3,139
Trainable params: 3,139
Non-trainable params: 0
```

圖 4. 模型建立

4.4 模型優化:

本專題使用正規化(Normalization)和 Dropout、L1、L2 正規化來對資料進行前處理，正規化目的主要是為了將資料集內分布的數據各點縮小至一定的範圍中，以便進行機器學習訓練時更能有效率的抓出各個數據之間的關係，而為了避免產生過度學習(Overfitting)的問題，本專題於神經網路模型設計上有使用 Dropout 的技術。Dropout 的原理為隨機捨棄一定比例的隱藏層輸出值，也可理解為隨機捨棄一定比例的隱藏層神經元。

Normalization	Dropout
	

L1 正規化的主要目的是重新做一次特徵的選擇，將一些重要的特徵留下，而 λ 越大則模型對於特徵的依賴性越低，此方法可以降低 overfitting 的產生。

L1 正規化損失函數 ：$\min_w[\sum_{i=1}^{N}(w^T x_i - y_i)^2 + \lambda \| w \|_1]$

L2 正規化的主要目的是將權重的數值縮小來減少對其的依賴性，而 λ 越大則模型對於特徵的依賴性越低，此方法可以降低 overfitting 的產生。

L2 正規化損失函數 ：$\min_w[\sum_{i=1}^{N}(w^T x_i - y_i)^2 + \lambda \| w \|_2^2]$

4.5 優化器選擇:

優化器為 Crossentropy，會使用這個優化器的原因是因為 Crossentropy 適合用來做分類問題的優化，關於如何呼叫該程式函數(如圖 5。

```
model.compile(keras.optimizers.Adam(), #優化器為0.001的Adam
              loss=keras.losses.CategoricalCrossentropy(), #損失函數為Crossentropy
              metrics=[keras.metrics.CategoricalAccuracy()]) #指標函數為Crossentropy
```

圖 5.Crossentropy python 程式碼

第五章 實驗結果與分析

5.1 資料前處理:

經過了研究發現，如果將 1 及 0 的數值與其他數值正規化則 1 及 0 的數值會變一模一樣，所以我們先將需要正規化的數值先進行讀取在做正規化，將資料列中的 Packet Loss Rate、Packet delay 讀取出來放入 data_Packet 內將 data_Packet 的內容顯示出來如下圖。

```
data_Packet = data.loc[:, :'Packet delay']
data_Packet.head(10)
```

]:

	Packet Loss Rate	Packet delay
0	0.000001	10
1	0.001000	100
2	0.000001	300
3	0.010000	100
4	0.010000	50
5	0.000001	10
6	0.010000	300
7	0.001000	50
8	0.001000	150
9	0.001000	150

將原始資料做正規化 Data_Packet 透過函數座 mean(平均值)的運算和做 std (標準差)之後再做正規化，下面再顯示出正規化完後的資料。

```
mean = data_Packet.mean()#利用.mean()函式計算 data_Packet 的平均值
std = data_Packet.std() #利用.std()函式計算 data_Packet 的標準差
normalization_data_Packet = (data_Packet - mean) / std#正規化data_Packet (公式 : (x - 平均值) / 標準差)
```

```
normalization_data_Packet.head()
```

	Packet Loss Rate	Packet delay
0	-0.708641	-0.979362
1	-0.478670	-0.132869
2	-0.708641	1.748227
3	1.593144	-0.132869
4	1.593144	-0.603143

之後透過 join 的方式將正規畫完成的資料放回資料表內，其結果如下圖。

```
data.drop(['Packet Loss Rate','Packet delay'], axis='columns', inplace=True)
data = data.join(normalization_data_Packet)
data.head()
```

	IoT	LTE/5G	GBR	Non-GBR	AR/VR/Gaming	Healthcare	Industry 4.0	IoT Devices	Public Safety	Smart City & Home	Smart Transportation	Smartphone	slice Type	Packet Loss Rate	Packet delay
0	1	0	0	1	0	0	0	0	1	0	0	0	3	-0.708641	-0.979362
1	0	1	1	0	1	0	0	0	0	0	0	0	1	-0.478670	-0.132869
2	0	1	0	1	0	0	0	0	0	0	0	1	1	-0.708641	1.748227
3	0	1	0	1	0	0	0	0	0	0	0	1	1	1.593144	-0.132869
4	1	0	0	1	0	0	0	0	0	1	0	0	2	1.593144	-0.603143

將 slice type 的資料讀取出來並做 one hot encoding 然後將產生出來的三個欄位分別命名為 eMBB、mMTC、URLLC 並顯示出其結果如下。

```
slice_one_hot = pd.get_dummies(data['slice Type'])
slice_one_hot.columns = ['eMBB','mMTC','URLLC']
slice_one_hot.head()
```

	eMBB	mMTC	URLLC
0	0	0	1
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0

之後將 one hot encoding 完的資料先合併回原本的資料表，並刪除原本的 slice type 的資料欄位，並顯示其結果。

```
slice_df_one_hot = slice_one_hot.astype('int64')
data_df = data.join(slice_df_one_hot)
data_df.drop('slice Type', axis='columns', inplace=True)
data_df.head()
```

LTE/5G	GBR	Non-GBR	AR/VR/Gaming	Healthcare	Industry 4.0	IoT Devices	Public Safety	Smart City & Home	Smart Transportation	Smartphone	Packet Loss Rate	Packet delay	eMBB	mMTC	URLLC
0	0	1	0	0	0	0	1	0	0	0	-0.708641	-0.979362	0	0	1
1	1	0	1	0	0	0	0	0	0	0	-0.478670	-0.132869	1	0	0
1	0	1	0	0	0	0	0	0	0	1	-0.708641	1.748227	1	0	0
1	0	1	0	0	0	0	0	0	0	1	1.593144	-0.132869	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1.503144	-0.603143	0	1	0

再來將資料讀入 x_train、y_train、x_val、y_val，之後將 y_train、y_val 進行 reshape((-1,3))將資料進行參數得綁定。

```
x_train = np.array(train_data.drop(['eMBB','mMTC','URLLC'], axis='columns')) #刪掉train_data輸入層的slice Type
#y_train =np.array(normalization_train_data.drop(['Packet loss Rate','Packet delay','IoT','LTE/5G','GBR','Non-GBR','AR/VR/Gam
y_train = np.array(train_data.loc[:, 'eMBB':])
y_train = y_train.reshape((-1,3))
x_val = np.array(val_data.drop(['eMBB','mMTC','URLLC'], axis='columns')) #刪掉train_data輸入層的slice Type
y_val = np.array(val_data.loc[:, 'eMBB':])
y_val = y_val.reshape((-1,3))
#y_val =np.array(normalization_val_data.drop(['Packet loss Rate','Packet delay','IoT','LTE/5G','GBR','Non-GBR','AR/VR/Gaming'
```

下面說明 reshape 內的參數顯示的內容假如說參數為(-1,3)則它會成為一個一維陣列、參數為(2,4)時則會變成一個二維陣列、參數為(2,2,2)時則會變成一個三維陣列如下圖。

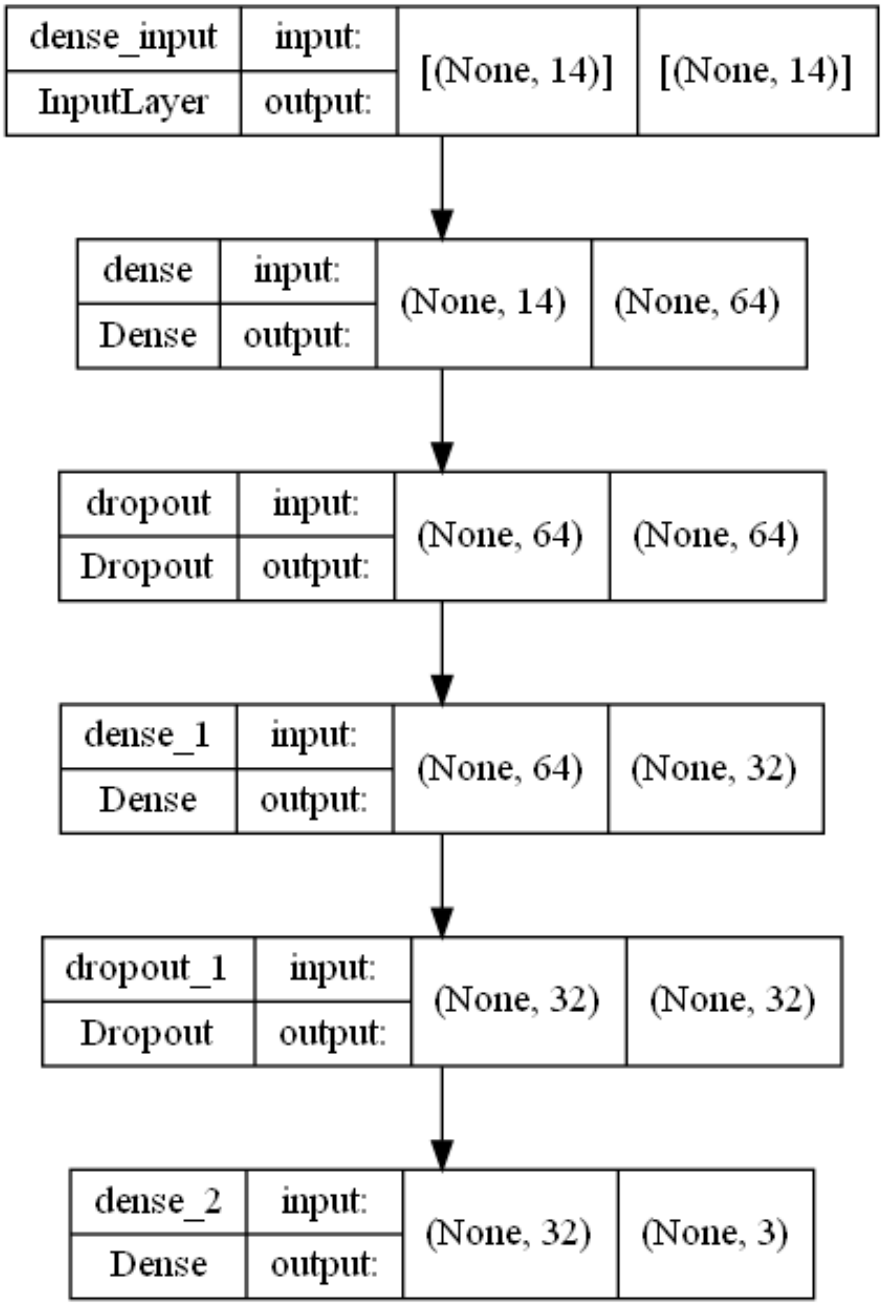
```
reshape((-1,3))
array([1,2,3])

reshape((2,4))
array([1,2,3,4],[5,6,7,8])

reshape((2,2,2))
array([[1,2],[3,4]],[[5,6],[7,8]])
```

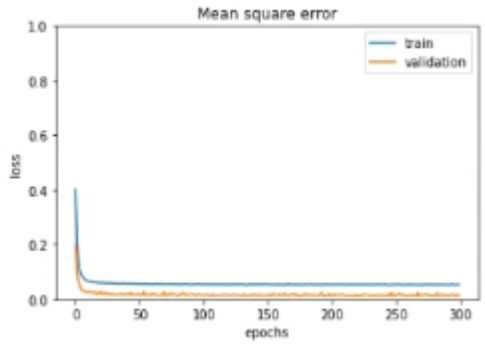
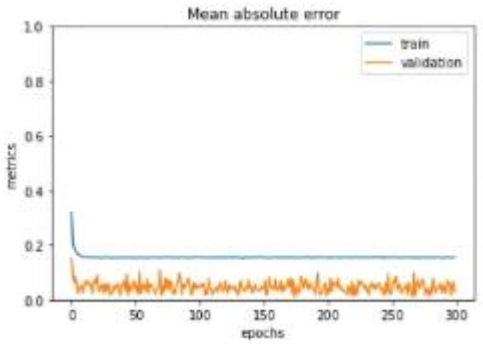
5.2 模型架構:

下圖為本次專題的模型 第一層有 14 個輸入並做了 L2 的正規化然後激活函數為 relu，而用 L2 正規化的原因將權重數值縮小，再來再進行 droup out 來隨機捨棄一定比例的隱藏層神經元，第三層 L1 的正規化然後激活函數為 relu，而用 L1 正規化的原因將較為突出的特徵顯示出來，之後再次進行 droup out 最後因為是這份數據主要要做的是多然分類，所以最後一層的激活函數使用的是 softmax。



5.3 結果與分析:

經過了多次的的測試主要分為兩種模型的的結果，第一種是沒有做 one hot encoding 和激活函數最後一層為 relu 結果如下表

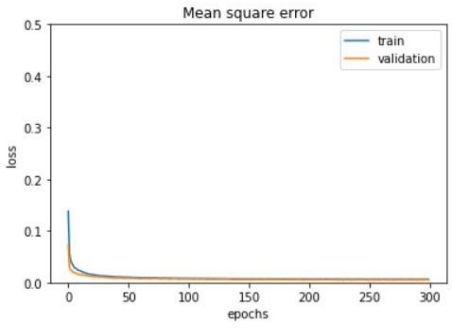
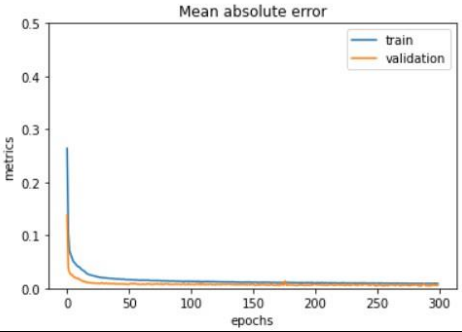
Mean square error	Mean absolute error
	


```
# 載入模型
model = keras.models.load_model('models/model1-1/Best-model-1.h5')

y_test = np.array(test_data['slice Type'])
# 標準化數據
test_data = (test_data - mean) / std
# 將輸入數據序列Numpy 格式
x_test = np.array(test_data.drop('slice Type', axis='columns'))
# 預測測試數據
y_pred = model.predict(x_test)
y_pred = np.reshape(y_pred * std['slice Type'] + mean['slice Type'], y_test.shape)
# 計算平均的誤差百分比
percentage_error = np.mean(np.abs(y_test - y_pred)) / np.mean(y_test) * 100
# 顯示誤差百分比
print("Model_1 Percentage Error: {:.2f}%".format(percentage_error))
```

```
Model_1 Percentage Error: 0.48%
```

下面結果為 slice type 做 one hot encoding，激活函數最後一層為 soft max，結果如下表

Mean square error	Mean absolute error
	


```
In [81]:
model3 = keras.models.load_model('models1/model3/Best-model3.h5')
x_test = np.array(test_data.loc[:, :'Packet delay'])
y_test = np.array(test_data.loc[:, 'eMBB':])
loss,accuracy = model3.evaluate(x_test, y_test)
print("Model-1: {}%".format(accuracy))
```

```
198/198 [==============================] - 0s 516us/step - loss: 0.0052 - mean_absolute_error: 0.0048
Model-1: 0.004845454357564449%
```

透過兩個不同模型優化的比對，可以發現有做 one hot encode 、 softmax 的不管是 Mean square error 或 Mean absolute error 還是損失值都都有大幅度的下降，所以由此可證在面對多元分類及數值大小並做 softmax 和 one hot encode 對於優化訓練出來的權重是非常有幫助的，並且在此次專題也了解到對於模型的優化不論是正規化或者 Dropout 都可以有效的解決過度學習的問題。

參考文獻

[1] Microsoft, “github,” Microsoft, 25 3 2023. [線上]. Available: https://github.com/microsoft/FLAML.

[2] 3GPP, “NR;Radio Resorce Control(RRC)procotol specigication(Release 17),” 於 *TS38.301v17.3.0*, 2022.

[3] Samsung, “Technical White Paper,” *Network Slicing*, p. 22, 2020.

[4] G. Dutta, “Network Slicing Recognition,” kaggle, 5 2022. [線上]. Available: https://www.kaggle.com/datasets/gauravduttakiit/network-slicing-recognition.