1. 目的：根據 200 名病患提供的腦部核磁共振造影的 3D 影像來預測其他 40 名病患是否患有腦瘤。

2. 資料和實作：

(1) 資料型態：共有 200 筆病患的數據資料，包含 T1 和 T2 兩種以不同成像方式獲取的 DICOM 影像，個別有 22 張，而每位病患擁有的幀數不一。因 MRI 影像資料型態為 3D，故在預處理上將 channel 數改為[T1 灰階,T2 灰階,平均灰階]，以下為實作方法和成果：

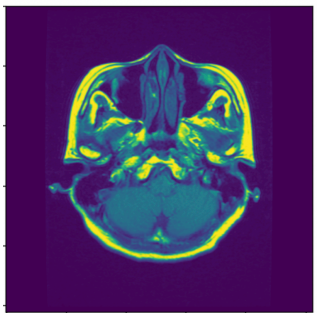
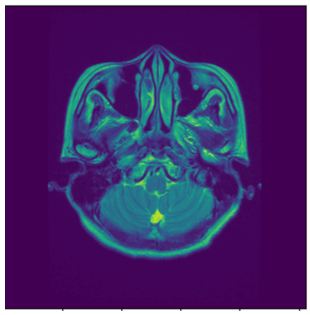
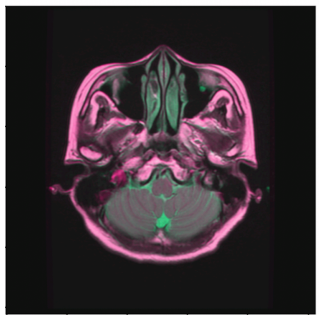
T1 的讀取方式，T2 亦同：

```
T1 = []
for i in train['ID']:
    path = 'C:\\Users\\renee\\DICOM2\\'+ i +'\\T1'
    os.chdir(path) #到指定資料夾
    x = os.listdir()
    data = [0]*len(x)
    for j in x:
        ds = dcmread(j)
        WC = ds.WindowCenter
        WW = ds.WindowWidth
        H = (int(ds.WindowCenter)*2 + int(ds.WindowWidth)) / 2
        L = H - int(ds.WindowWidth)
        arr = np.array(ds.pixel_array)
        arr[arr > H] = H
        arr[arr < L] = L
        arr = 255*((arr-L)/int(WW))
        data[int(ds.InstanceNumber)-1] = arr
    data = np.array(data)
    x_np = torch.from_numpy(data)
    x_np = x_np[None, None, :]
    x_np = torch.nn.functional.interpolate(x_np, size= [22,512,512], scale_factor=None, mode='trilinear')
    x_np = torch.reshape(x_np, (22,512,512))
    x_np = x_np.numpy()
    T1.append(x_np)
```

將 T1、T2 和[(T1+T2)/2]堆疊起來：

```
x = T1[0]
y = T2[0]
z = (T1[0]+T2[0])/2
image = np.expand_dims(x,axis=3)
i2 = np.expand_dims(y,axis=3)
i3 = np.expand_dims(z,axis=3)
image = np.concatenate((image,i2,i3),axis=-1)
image = image / 255

plt.imshow(image[2])
plt.show()
```

T1 和 T2 以及將 T1、T2 和[(T1+T2)/2]堆疊起來的 3D 造影：

T1	T2	3D 造影
		

(2)資料擴增和模型：

此次所展示的模型為 late fusion model，這個模型輸入的資料型態為 2D 影像，故在特徵上會少了 depth 這部分的資訊，故原始資料的 shape 為 (width, height, channel) = (512, 512, 3)，意即將每個病人的腦部造影分別送入模型中，而非將其堆疊起來。而因使用的環境為 Colab，因此將每張圖片 resize 為 350 * 350，為防止圖片太小，並將圖片放大 1.5 倍，使腫瘤能更加明顯，Colab 的記憶體空間不夠存放 22 個 model 的參數，故在 feature selection 上，先將圖片依序排好後，每個病人皆只取六張較看得出腫瘤的影像，分別為第 11~15 張影像，並分成訓練和驗證資料分別存入 F1~F5 的矩陣中，如下所示：

train data:

```
F1_train = []
F2_train = []
F3_train = []
F4_train = []
F5_train = []
for i in range(len(X_train)):
  for j in range(11,16):
    if(j==11):
      F1_train.append(X_train[i][j])
    if(j==12):
      F2_train.append(X_train[i][j])
    if(j==13):
      F3_train.append(X_train[i][j])
    if(j==14):
      F4_train.append(X_train[i][j])
    if(j==15):
      F5_train.append(X_train[i][j])
```

```
F1_train = np.array(F1_train)
F2_train = np.array(F2_train)
F3_train = np.array(F3_train)
F4_train = np.array(F4_train)
F5_train = np.array(F5_train)
```

```
F1_train.shape
```

```
(112, 350, 350, 3)
```

validation data:

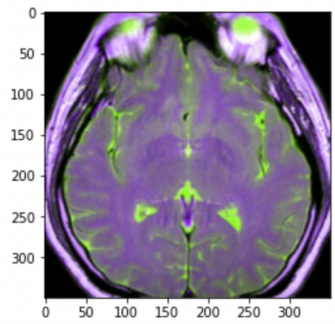
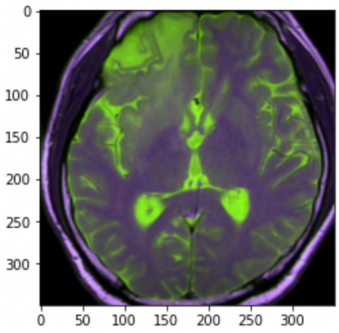
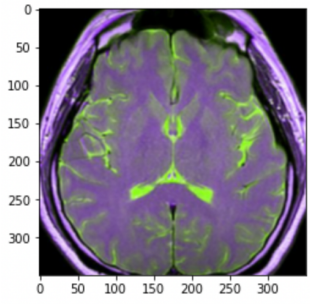
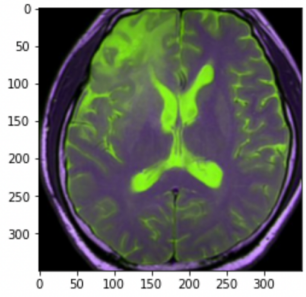
```
F1_valid = []
F2_valid = []
F3_valid = []
F4_valid = []
F5_valid = []
for i in range(len(X_valid)):
  for j in range(11,16):
    if(j==11):
      F1_valid.append(X_valid[i][j])
    if(j==12):
      F2_valid.append(X_valid[i][j])
    if(j==13):
      F3_valid.append(X_valid[i][j])
    if(j==14):
      F4_valid.append(X_valid[i][j])
    if(j==15):
      F5_valid.append(X_valid[i][j])
```

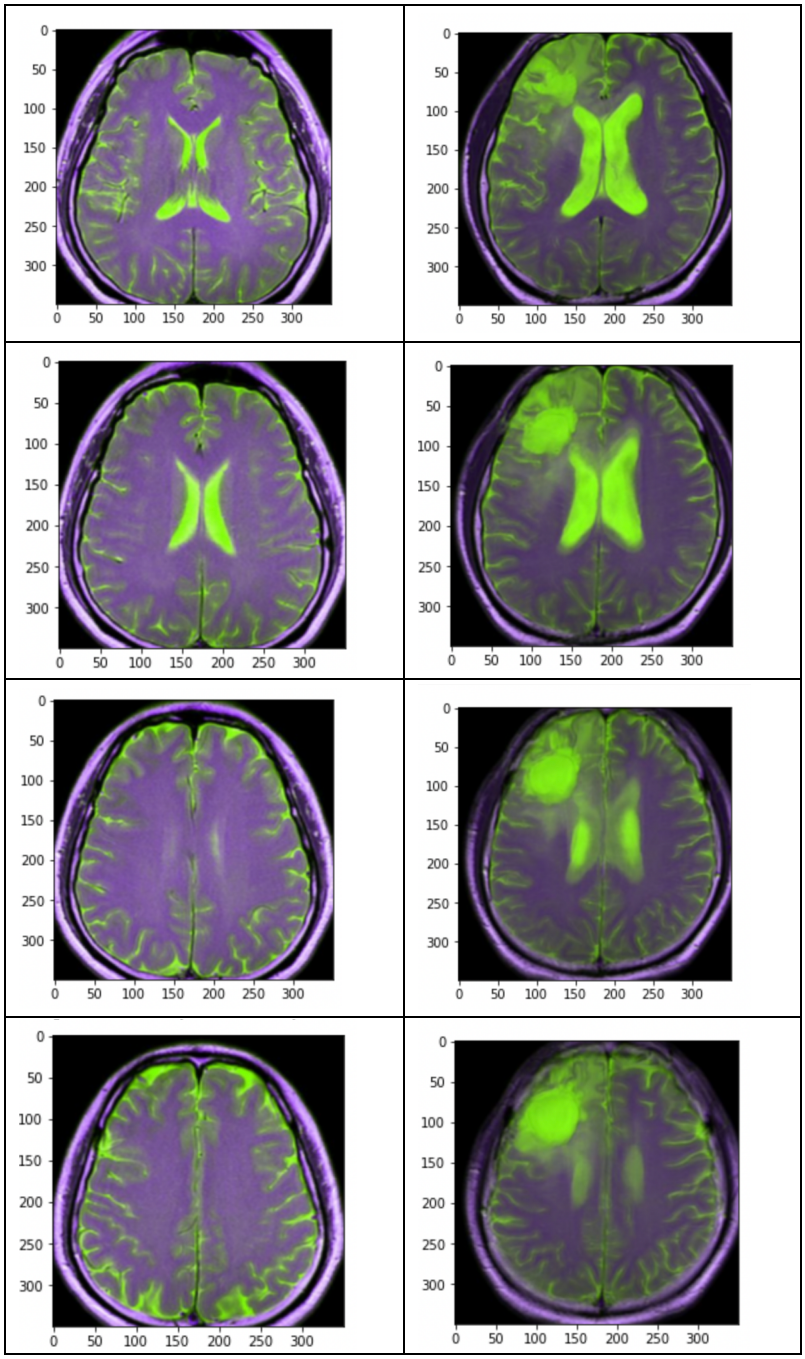
```
F1_valid = np.array(F1_valid)
F2_valid = np.array(F2_valid)
F3_valid = np.array(F3_valid)
F4_valid = np.array(F4_valid)
F5_valid = np.array(F5_valid)
```

```
F1_valid.shape
```

```
(48, 350, 350, 3)
```

影像如下所示：

Normal	Tumor
	
	



而在 Data augmentation 方面，每一個病人的 augmentation 需一致，故將同一個病人的六張圖片包起來進行 augmentation 後再將六張照片 concatenate 起來，而 validation data 的部分僅將六張照片 concatenate 其餘不做任何處理，以 train data 為例，如下所示：

Data Augmentation

```
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import Sequence
class MultipleImageGenerator(Sequence):
  def __init__(self, x1 ,x2 ,x3, x4, x5, y,batch_size):
    self.generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.1,
    brightness_range=[0.5, 1.25],
    horizontal_flip=True,
    )
    self.gen_x1 = self.generator.flow(x1, y, batch_size = batch_size, seed=1)
    self.gen_x2 = self.generator.flow(x2, y, batch_size = batch_size, seed=1)
    self.gen_x3 = self.generator.flow(x3, y, batch_size = batch_size, seed=1)
    self.gen_x4 = self.generator.flow(x4, y, batch_size = batch_size, seed=1)
    self.gen_x5 = self.generator.flow(x5, y, batch_size = batch_size, seed=1)
  def __len__(self):
    return self.gen_x1.__len__()

  def __getitem__(self,index):
    x1_batch, y_batch = self.gen_x1.__getitem__(index)
    x2_batch, y_batch = self.gen_x2.__getitem__(index)
    x3_batch, y_batch = self.gen_x3.__getitem__(index)
    x4_batch, y_batch = self.gen_x4.__getitem__(index)
    x5_batch, y_batch = self.gen_x5.__getitem__(index)
    x_batch = [x1_batch, x2_batch, x3_batch, x4_batch, x5_batch]
    return x_batch, y_batch
```

在模型方面，因 VGG16 模型太大，故自行定義了 model，由於有六張圖片，因此先分別定義 6 個 model，取 vgg16 模型中的前三個 block，因環境關係，每一個 block 的 filter 皆等比例縮小，且在第二個 block 中加入 Batch Normalization 防止 overfitting，且每一個 model 最後的 output 都使用 flatten 將參數變成一維，以 model 1 為例，結構如下所示，其餘 model2~5 皆相同：

```
#Late fusion--> Each patient only concatenate 5 feature map
#model1
#conv2D block 1
base_mod1 = Sequential(name = 'F1')
base_mod1.add(Conv2D(filters=64,kernel_size=(3,3),input_shape=(350,350,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(Conv2D(filters=64,kernel_size=(3,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(MaxPool2D())
#conv2D block 2
base_mod1.add(Conv2D(filters=64,kernel_size=(3,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(Conv2D(filters=64,kernel_size=(3,3),padding='same'))
base_mod1.add(BatchNormalization())
base_mod1.add(Activation('relu'))
base_mod1.add(MaxPool2D((2, 2), strides=(2, 2)))
#conv2D block 3
base_mod1.add(Conv2D(filters=128,kernel_size=(3,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(Conv2D(filters=128,kernel_size=(3,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(Conv2D(filters=128,kernel_size=(3,3),padding='same'))
base_mod1.add(Activation('relu'))
base_mod1.add(MaxPool2D((2, 2), strides=(2, 2)))
base_mod1.add(Flatten())
y1 = base_mod1.output
base_mod1.summary()
```

```
Model: "F1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_70 (Conv2D)           (None, 350, 350, 64)      1792

 activation_70 (Activation)   (None, 350, 350, 64)      0

 conv2d_71 (Conv2D)           (None, 350, 350, 64)      36928

 activation_71 (Activation)   (None, 350, 350, 64)      0

 max_pooling2d_30 (MaxPoolin  (None, 175, 175, 64)      0
 g2D)

 conv2d_72 (Conv2D)           (None, 175, 175, 64)      36928

 activation_72 (Activation)   (None, 175, 175, 64)      0

 conv2d_73 (Conv2D)           (None, 175, 175, 64)      36928

 batch_normalization_12 (Bat  (None, 175, 175, 64)      256
 chNormalization)

 activation_73 (Activation)   (None, 175, 175, 64)      0

 max_pooling2d_31 (MaxPoolin  (None, 87, 87, 64)        0
 g2D)

 conv2d_74 (Conv2D)           (None, 87, 87, 128)       73856

 activation_74 (Activation)   (None, 87, 87, 128)       0

 conv2d_75 (Conv2D)           (None, 87, 87, 128)       147584

 activation_75 (Activation)   (None, 87, 87, 128)       0

 conv2d_76 (Conv2D)           (None, 87, 87, 128)       147584

 activation_76 (Activation)   (None, 87, 87, 128)       0

 max_pooling2d_32 (MaxPoolin  (None, 43, 43, 128)       0
 g2D)

 flatten_12 (Flatten)         (None, 236672)            0

=================================================================
Total params: 481,856
Trainable params: 481,728
Non-trainable params: 128
```

再將五個 model 的 output(feature maps) concatenate 形成最後完整的模型，且再 activation function 前再加上 BN 層，和最後一層 sigmoid 前加入 regularization 防止過擬合，而因為參數較多，所以 batch size 僅設為 8，learning rate 一樣設為 0.0001，完整模型如下所示：

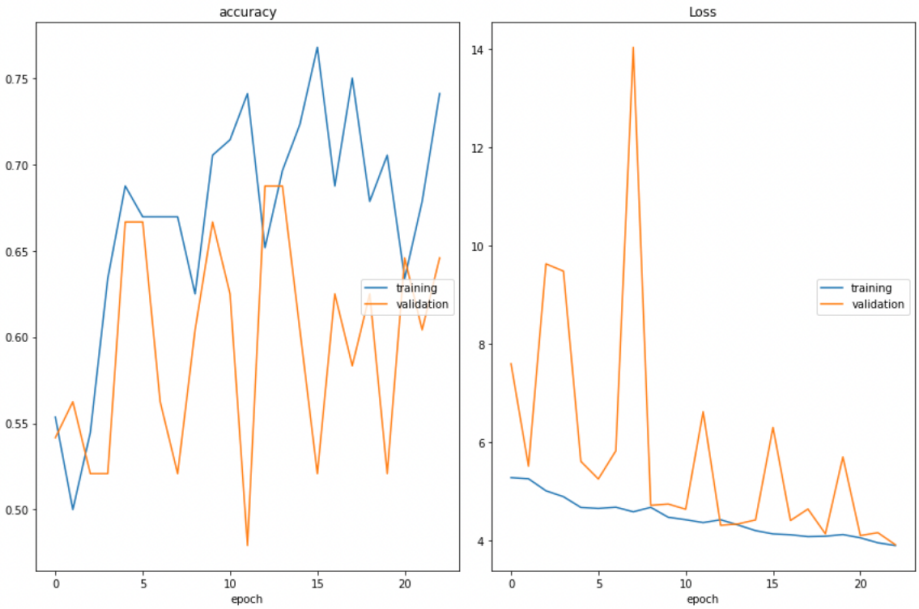
```
#full model
x = concatenate([y1,y2,y3,y4,y5])
x = Sequential()(x)
x = Flatten()(x)
x = Dense(128)(x)
x =BatchNormalization()(x)
x = Dense(64,activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(32,kernel_regularizer=regularizers.l2(0.1),activation="relu")(x)
predictions = Dense(1,activation='sigmoid')(x)

model = Model(inputs=[base_mod1.input,base_mod2.input,base_mod3.input,base_mod4.input,base_mod5.input],outputs=predictions
model.compile(loss='binary_crossentropy',optimizer=optimizers.Adam(learning_rate=0.0001) ,metrics=['accuracy'])
model.summary()
```

只截取完整模型的架構，如下所示：

```
flatten_12 (Flatten)           (None, 236672)       0           ['max_pooling2d_32[0][0]']

flatten_13 (Flatten)           (None, 236672)       0           ['max_pooling2d_35[0][0]']

flatten_14 (Flatten)           (None, 236672)       0           ['max_pooling2d_38[0][0]']

flatten_15 (Flatten)           (None, 236672)       0           ['max_pooling2d_41[0][0]']

flatten_16 (Flatten)           (None, 236672)       0           ['max_pooling2d_44[0][0]']

concatenate_2 (Concatenate)    (None, 1183360)      0           ['flatten_12[0][0]',
                                                                 'flatten_13[0][0]',
                                                                 'flatten_14[0][0]',
                                                                 'flatten_15[0][0]',
                                                                 'flatten_16[0][0]']

sequential_2 (Sequential)      multiple             0           ['concatenate_2[0][0]']

flatten_17 (Flatten)           (None, 1183360)      0           ['sequential_2[0][0]']

dense_8 (Dense)                (None, 128)          151470208   ['flatten_17[0][0]']

batch_normalization_17 (BatchN  (None, 128)         512         ['dense_8[0][0]']
ormalization)

dense_9 (Dense)                (None, 64)           8256        ['batch_normalization_17[0][0]']

dropout_2 (Dropout)            (None, 64)           0           ['dense_9[0][0]']

dense_10 (Dense)               (None, 32)           2080        ['dropout_2[0][0]']

dense_11 (Dense)               (None, 1)            33          ['dense_10[0][0]']

==================================================================================================
Total params: 153,890,369
Trainable params: 153,889,473
Non-trainable params: 896
__________________________________________________________________________________________________
```

訓練過程如下所示：

```
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    patience=10
)
from livelossplot import PlotLossesKeras
plotlosses=PlotLossesKeras()
epochs = 30
model.fit(train,
          epochs = epochs,
          validation_data = valid,
          callbacks = [plotlosses,early_stopping])
```




```
accuracy
        training                (min:    0.500, max:    0.768, cur:    0.741)
        validation              (min:    0.479, max:    0.688, cur:    0.646)
Loss
        training                (min:    3.902, max:    5.284, cur:    3.902)
        validation              (min:    3.920, max:   14.034, cur:    3.920)
14/14 [==============================] - 435s 31s/step - loss: 3.9020 - accuracy: 0.7411 - val_loss: 3.9200 - val_accuracy: 0.6458
<keras.callbacks.History at 0x7f05fbd498d0>
```

訓練結果部分如下所示：

	Disease0	Disease1	Disease
0	[0.5023083090782166]	[0.49769169092178345]	0
1	[0.8025899529457092]	[0.19741006195545197]	0
2	[0.039984047412872314]	[0.9600159525871277]	1
3	[0.04120403528213501]	[0.958795964717865]	1
4	[0.3778076171875]	[0.6221923828125]	1
5	[0.4028193950653076]	[0.5971806049346924]	1
6	[0.5754474401473999]	[0.4245525598526001]	0
7	[0.014241218566894531]	[0.9857587814331055]	1
8	[0.7289137840270996]	[0.271086186170578]	0
9	[0.6620055437088013]	[0.33799442648887634]	0
10	[0.8249248266220093]	[0.1750752031803131]	0
11	[0.8470313549041748]	[0.1529686450958252]	0

部分實際結果如下所示：

patient	Late	Correct
1	0	0
2	0	0
3	1	1
4	1	1
5	1	0
6	1	1
7	0	1
8	1	1
9	0	1
10	0	0
11	0	0
12	0	1

準確度為 24/40 = 62.5%，推測結果因 Latefusion model 因捨棄 depth 的資訊，加上因記憶體空間不夠，只取每個病人 22 張圖片中的中間 6 張，因此遺失了很多資訊。