Assignment 3

COVID-19 30-day Mortality Prediction from CXR Report

106062314 蔡政諺

● Code Link

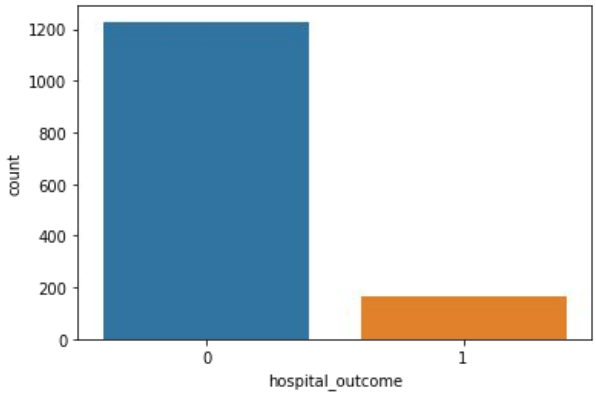
https://colab.research.google.com/drive/1lQ3eGjsbrJS8YVV4Drq5l_fxJda6mRfj

● Summary

- 使用 Keras 實作 CNN model。
- 移除 outlier，避免 model 學習錯誤的特徵。
- 將影像 rescale 為 64×64×1，減少 input dimension。
- 使用 ImageDataGenerator 做 data augmentation。
- 處理 mixed data，完成 bonus task。
- 在 validation data 達到 f1 score = 0.5122 的成效。

● Code Descriptions

首先從 **cxr_label_train.csv** 讀取 patient id 與 label，並且因為 patient id 112 明顯是 outlier，我將其移除。此時可以看出 label 的分布非常不平均。



接著依照 patient id 的順序，從 **IML_CXR_TRAIN** 將影像讀出，設定為讀取 grayscale image、rescale 為 64×64、並 normalize 為 0 到 1 的值。原本我試過將 image size 設為 320×320，但不僅 training 速度會變慢，且直覺上在這麼大張的圖片上使用相對小的 filter

size，model 應該會想去捕捉枝微末節的特徵，理論上並不會達到太好的效果，而我實際嘗試過後也是如此。

接著從 hw2 的 **hm_hospitales_covid_structured_30d_train.csv** 依序將 numeric data 讀出，並做 preprocess。Preprocess 包含對 missing value 做插值、將 class value 轉換成 one-hot、把用不到的 column 刪掉。經過處理後，共會有 50 個 features。

此時我們有 (1392, 64, 64, 1) 的 image data (1 代表 color channel)、(1392, 50) 的 numeric data、以及 (1392, 1) 的 label，所有資料的型別都被轉換為 numpy array。

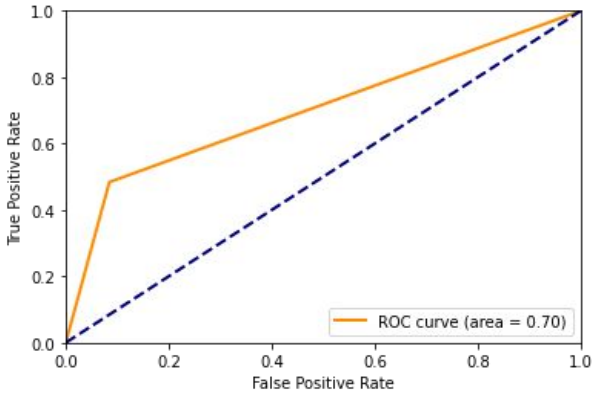
```
print("image data   :", data.shape)
print("numeric data :", num_data.shape)
print("label        :", label.shape)
```

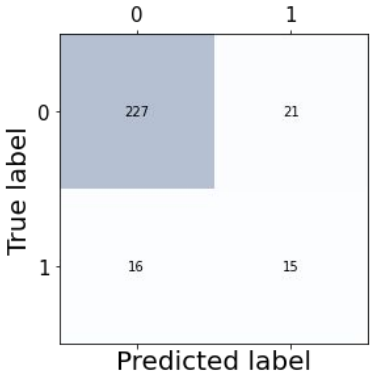
```
image data   : (1392, 64, 64, 1)
numeric data : (1392, 50)
label        : (1392, 1)
```

接著便可呼叫 train_test_split，將資料切成 training data 與 validation data。由於 label 為 0 與 1 的分布非常不平均，此處我對 training data 做 data augmentation。這裡我首先將 label 為 1 的 training data 取出，並使用 ImageDataGenerator，取出 label 為 1 的 training data 做 random augmentation (rotation、shift、horizontal flip)，同時從 label 為 1 的 numeric data 隨機挑一筆 append，直到 label 為 0 與 1 的 training data 一樣多。

接著就可以建立 keras model。我使用 convolutional layer、maxpooling layer、batch normalization layer、最後接上 dense layer。因為是做 binary classification，Loss function 使用 binary crossentropy，optimizer 則選用 Adam。

```
def buildModel():
  img_data = keras.layers.Input(shape=(64, 64, 1))
  x = keras.layers.Conv2D(32, 6, strides=2, input_shape=(64, 64, 1), activation='relu')(img_data)
  x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Conv2D(32, 3, strides=2, activation='relu')(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Conv2D(32, 3, strides=1, activation='relu')(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Flatten()(x)
  x = keras.layers.Dense(units=128, activation='relu')(x)
  x = keras.layers.Dense(units=1, activation='sigmoid')(x)
  return keras.Model(inputs=img_data, outputs=x)
```



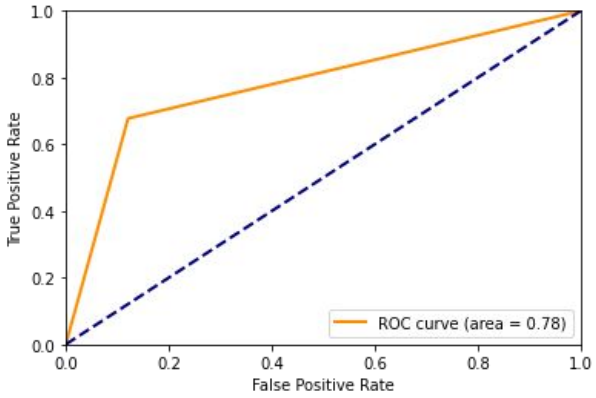


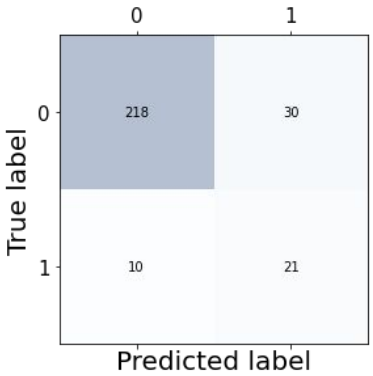
我在 validation data 達到最好的 performance 為 f1 score = **0.4478**，我將這次當作最終結果輸出。

- **Code Descriptions (Bonus)**

在 bonus 中，必須同時處理 image data 與 numeric data，因此我建了可以輸入兩組 input 的 keras model。對 image data 使用的 layer 與上面相同，在 numeric data 我則使用了三層的 dense layer，以及 batch normalization。兩者分別做完 feature extraction 後，可以 concatenate 成一層 flattened layer。接著再接一層 units=128 的 hidden layer，以及一層 units=1 的 output layer。Loss function 和 optimizer 與前者相同。

```
def buildBonusModel():
  img_data = keras.layers.Input(shape=(64, 64, 1))
  num_data = keras.layers.Input(shape=(50))

  # image data extraction
  x = keras.layers.Conv2D(32, 6, strides=2, input_shape=(64, 64, 1), activation='relu')(img_data)
  x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Conv2D(32, 3, strides=2, activation='relu')(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Conv2D(32, 3, strides=1, activation='relu')(x)
  x = keras.layers.BatchNormalization()(x)
  x = keras.layers.Flatten()(x)
  x = keras.Model(inputs=img_data, outputs=x)

  # numeric data extraction
  y = keras.layers.Dense(units=32, activation="relu")(num_data)
  y = keras.layers.BatchNormalization()(y)
  y = keras.layers.Dense(units=16, activation="relu")(y)
  y = keras.layers.BatchNormalization()(y)
  y = keras.layers.Dense(units=8, activation="relu")(y)
  y = keras.layers.BatchNormalization()(y)
  y = keras.Model(inputs=num_data, outputs=y)

  # fully connected layer
  combined = keras.layers.concatenate([x.output, y.output])
  z = keras.layers.Dense(units=128, activation="relu")(combined)
  z = keras.layers.Dense(units=1, activation="sigmoid")(z)

  return keras.Model(inputs=[x.input, y.input], outputs=z)
```





使用 bonus model 時，我在 validation data 達到最好的 performance 為 f1 score = **0.5122**，我將這次當作最終結果輸出。

- **How to Use the Model File**

[Hw3-64x64.ipynb - Colaboratory (google.com)]

首先將 model 的 h5 檔讀取出來。

```
model = keras.models.load_model(dir + "output/106062314_HW3_Model.h5")

bonusModel = keras.models.load_model(dir + "output/Bonus_106062314_HW3_Model.h5")
```

將 testing data 的 patient id 排序，排好後依序從 **IML_CXR_TEST** 將影像讀出， rescale 為 64×64，並 normalize 為 0 到 1 的值。

```
test_patient_id = []
for filename in os.listdir(dir + "IML_CXR_TEST/"):
  test_patient_id.append(int(filename[:-4]))
test_patient_id.sort()

test_data = []
for id in tqdm(test_patient_id):
  filepath = dir + "IML_CXR_TEST/" + str(id) + ".jpg"
  img = image.load_img(filepath, grayscale=True, target_size=(64, 64))
  img = np.array(img)
  test_data.append(img)
test_data = np.array(test_data).reshape(-1, 64, 64, 1)

# normalization
test_data = test_data / 255.0
```

依序從 hw2 的 **fixed_test.csv** 將 numeric data 讀出，並做與 training 相同的 preprocess。

```python
def getNumericData(filepath, patient_id):
  df = pd.read_csv(filepath)

  # impute missing values
  df = df.fillna(df.mean())

  # encode class input to one-hot
  df['sex'] = df['sex'].map({'MALE': 0, 'FEMALE': 1})
  ed_diagnosis_list = ['sx_breathing_difficulty', 'sx_flu', 'sx_fever', 'sx_cough', 'sx_others']
  for element in ed_diagnosis_list:
    df[element] = df['ed_diagnosis'].map(lambda x: 1 if x == element else 0)

  num_patient_id = np.array(df['PATIENT ID']).tolist()

  # drop useless information
  df = df.drop(['PATIENT ID', 'admission_datetime', 'ed_diagnosis'], axis=1)

  # convert dataframe to numpy array
  num_data_raw = np.array(df)

  # extract data with patient_id
  num_data = []
  for id in patient_id:
    row = num_patient_id.index(id)
    num_data.append(num_data_raw[row])
  num_data = np.array(num_data)
  return num_data

test_num_data = getNumericData(filepath=dir+"fixed_test.csv", patient_id=test_patient_id)

print("image data   :", test_data.shape)
print("numeric data :", test_num_data.shape)
```

此時我們有 (457, 64, 64, 1) 的 image data 與 (457, 50) 的 numeric data。

```python
print("image data   :", test_data.shape)
print("numeric data :", test_num_data.shape)
```

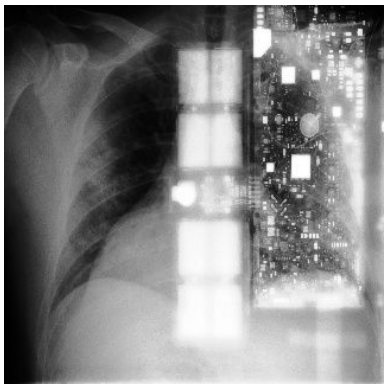
```
image data   : (457, 64, 64, 1)
numeric data : (457, 50)
```

接著就可以呼叫 predict，預測 testing data 的結果，並將機率 round 為整數，若輸出 0 代表預測該病人生存，輸出 1 代表預測該病人死亡。

```python
predict_label = model.predict(test_data).round()
predict_label = np.array(predict_label)

# save predictions
df = pd.DataFrame(predict_label)
df.columns = ['hospital_outcome']
df.index = test_patient_id
df.to_csv(dir + "output/106062314.csv")

predict_label = bonusModel.predict([test_data, test_num_data]).round()
predict_label = np.array(predict_label)

# save predictions
df = pd.DataFrame(predict_label)
df.columns = ['hospital_outcome']
df.index = test_patient_id
df.to_csv(dir + "output/Bonus_106062314.csv")
```

- **Discussions and Feedbacks**

這次作業的實作中，我一開始做出的 baseline 約為 **f1 score=0.2** 左右。為改善 model 的 performance，我嘗試在 preprocess 的部分做額外處理。首先，由於 **patient id 112** 的 image 過於不自然，故將其移除，避免 model 學習錯誤的特徵。



接著，若什麼 preprocess 都不做的話，input dimension 為 320×320×3。但若要處理這麼大維度的 input data，我們使用的 model 架構自然會大上許多，即需要更多的 parameter。但受限於樣本數的不足，**訓練太複雜的 model 可能會導致 overfitting**。這次作業中，由於輸入影像都是黑白影像，color channel 用 1 就可以表示；另一方面一張 320×320 的影像 rescale 成 64×64 後，其實在人眼觀察上並不會有太大的影響，因此處理 image data 的時候，我**將每張 320×320×3 的影像都 downscale 成 64×64×1**。這麼做可以在不損失過多 information 的情況下，讓 model architecture 相對不那麼複雜，一方面加速 training 的進行 (一個 epoch 原本約需 50 秒，降維後約只需 2 秒)，一方面也避免因為 model architecture 太複雜，導致根本 train 不起來。

```
Epoch 1/20                                          Epoch 1/20
31/31 [==============================] - 51s 2s/step 31/31 [==============================] - 4s 79ms/step
Epoch 2/20                                          Epoch 2/20
31/31 [==============================] - 48s 2s/step 31/31 [==============================] - 2s 71ms/step
Epoch 3/20                                          Epoch 3/20
31/31 [==============================] - 49s 2s/step 31/31 [==============================] - 2s 64ms/step
Epoch 4/20                                          Epoch 4/20
31/31 [==============================] - 46s 1s/step 31/31 [==============================] - 2s 66ms/step
Epoch 5/20                                          Epoch 5/20
31/31 [==============================] - 46s 2s/step 31/31 [==============================] - 2s 64ms/step
```

除此之外，因為這次處理的資料是 imbalanced data，若直接餵給 model，會導致 model 學習 label 為 0 的資料成效遠大於 label 為 1 的資料。有時候 model 甚至會為了 fit label 為 0 的資料，將所有 data 都 predict 為 0。因此我**使用 ImageDataGenerator 做 data augmentation**。詳細的作法是跑一個 for loop，每次從 label 為 1 的 image data 中取出一筆資料做 random augmentation，接著 append 到 training data 中；同時我會從 label 為 1

的 numeric data 中隨機取出一筆資料做 pairing，希望**讓 training data 在維度空間中分布到更多原本空缺的地方，使 model 更 robust**。

另外，在 model architecture 的設計上，我除了第一次 convolution 後面有接 max pooling，其餘兩次 convolution 都沒有使用，會這樣做是因為 **max pooling 的目的在於降維**，但相對地有可能將重要的 feature 丟掉，而且最初我已經做過降維，因此決定減少 max pooling 的使用。我也在 model 中加入了 batch normalization，這樣 feature 的分布才能回到中心，改善 training 的穩定性，且不會在最後偏移導致全部 predict 為 0 或 1。

實作上述幾個步驟後，model performance 平均約可達到 **f1 score=0.35 至 0.45** 左右，bonus 則可以達到 **f1 score=0.5** 左右，原本 precision 與 recall 會一大一小，現在兩者則能變得更對稱。

以往我只有使用過例如 MNIST、CIFAR-10 等資料量大且維度小的 dataset，而且這些 dataset 中，每個 class 的特徵都相當具代表性 (例如貓、狗的特徵就會相差甚遠)，因此 classification 非常好做。寫完這次作業，發現機器學習在 real task 下真的充滿許多挑戰，不僅不同 labal 之間的特徵差異不是那麼明顯，處理**樣本數不足、資料不平均、離群值**等等也是影響 performance 的關鍵，讓我深刻體會到資料處理的重要性。