인 공 지 능

H W 2

과 목 명: 인공지능
학 번: 201501428
학 과: 컴퓨터공학
이 름: 김 태 송

목 차

1. Dataset description

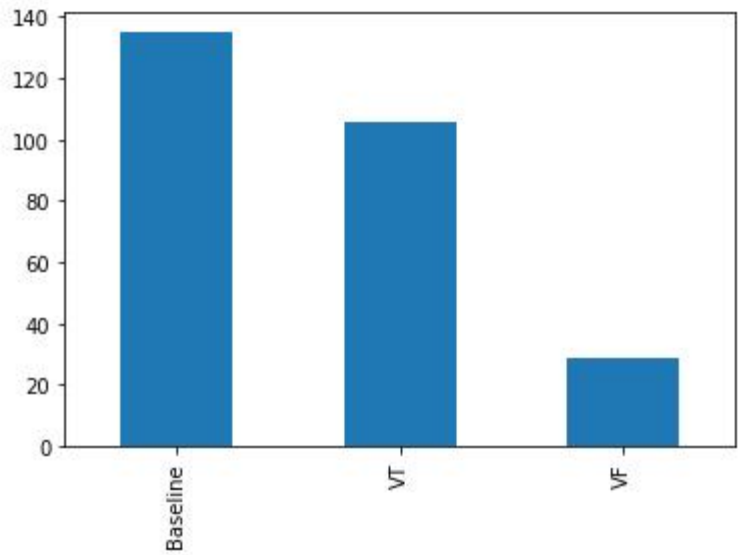
우선 케라스의 각종 모듈과 파이썬의 라이브러리를 import해서 사용할 패키지를 불러왔다. 심전도 데이터 csv 파일을 읽어와서 데이터의 구성이 어떻게 되는지 알아보기 위해서 우선 판다스를 이용해서 csv 파일을 읽어온다.

```
# 0 사용할 패키지 불러오기
from keras.models import Sequential # 케라스의 Sequential()을 임포트
from keras.layers import Dense, Activation # 케라스의 Dense()를 임포트
from keras.utils import np_utils
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# 판다스로 csv파일을 읽어오기
df= pd.read_csv("/content/drive/My Drive/Colab Notebooks/heartbeat_dat
asets.csv")
# csv 파일이 잘 읽히는지 확인
print(df.values)
# 데이터 파악
print(df.values.shape)
# 데이터는 총 270개 샘플로 구성되어져 있으며 988개의 열로 구성
print(df)
# 각 샘플의 인덱스는 986개의 시간별 심장박동률의 열과 심장의 상태를 나타내
는 1개의 열이 있다.
# Class열은 몇 가지 데이터로 구성되어져있는지 모든 데이터 종류출력
print("Class 종류:", df["Class"].unique(), sep="\n")
# Class열의 심장 데이터 구성 비율을 파악
df['Class'].value_counts().plot(kind='bar')
```

```
[[3 970 970 ... 950 950 'Baseline']
 [3 730 760 ... 240 240 'VT']
 [3 620 940 ... 700 700 'Baseline']
 ...
 [8079 820 490 ... 270 280 'VT']
 [8096 1080 1070 ... 1200 1170 'Baseline']
 [8096 740 740 ... 320 290 'VT']]
(270, 988)
     Sample_number    X1    X2    X3    X4  ...  X983  X984  X985  X986     Class
0                3   970   970   950   970  ...   940   950   950   950  Baseline
1                3   730   760   740   750  ...   230   250   240   240        VT
2                3   620   940   780   780  ...   710   710   700   700  Baseline
3                3   920   920   920   910  ...   280   280   280   280        VT
4                3   870   880   870   890  ...   860   850   850   850  Baseline
..             ...   ...   ...   ...   ...  ...   ...   ...   ...   ...       ...
265           8079  1230  1160  1130  1160  ...   270   270   280   270        VT
266           8079  1050  1030  1050  1060  ...  1050  1070  1080  1070  Baseline
267           8079   820   490  1140   820  ...   280   270   270   280        VT
268           8096  1080  1070  1040  1050  ...  1160  1190  1200  1170  Baseline
269           8096   740   740   590   920  ...   320   280   320   290        VT
```

데이터프레임 df라는 변수에 저장을 해서 데이터가 잘 출력이 되는지 확인을 했다. 코드를 실행하면 위의 그림처럼 csv 파일 데이터가 잘 읽힌 것을 볼 수 있고 데이터의 전체적인 구성을 확인하기 위해 행과 열의 개수인 shape를 출력했다. 데이터를 보면 Sample_number라는 표본과 X1~X986 심전도, 심장의 상태를 나타내는 Class 총 988개의 열이 있다. 이 중에서 우리는 심전도를 가지고 Class의 값에 따른 판단을 하고 싶은 것이 목적이기 때문에 Class열의 데이터의 종류를 확인하기 위해서 `df["Class"].unique(), sep="\n"` 를 이용해 종류가 몇 개인지를 파악했다. matplotlib을 이용해서 아래와 같이 그래프를 그려서 분포를 확인했다. 심장의 상태는 총 3가지가 있고 Baseline과 VT VF의 비율이 일정하지 않은 데이터셋인 것을 확인할 수 있다.

```
[270 rows x 988 columns]
Class 종류:
['Baseline' 'VT' 'VF']
<matplotlib.axes._subplots.AxesSubplot at 0x7ff843ceb198>
```



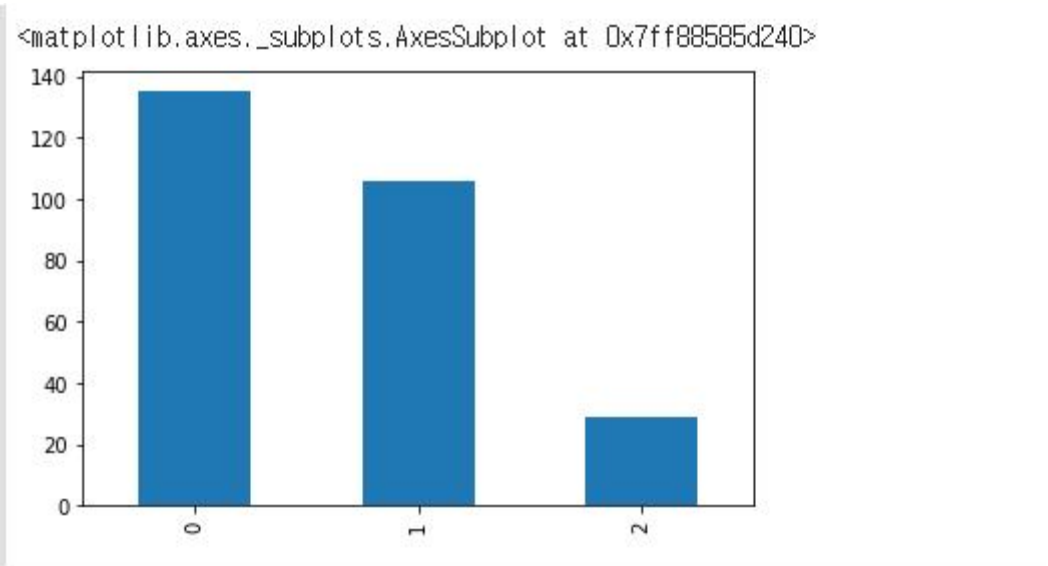
2. My idea

위의 데이터셋을 가지고 시간에 따른 심전도가 986개이고 심장의 상태의 종류가 3가지인 것을 확인 할 수 있었다. 따라서 모델을 만들 때 입력층에는 심전도 데이터 986개를 입력하는 입력이 있기 때문에 input_shape =(986,1)로 설정을 하고 마지막 출력층에서는 심장의 상태 3가지를 분류할 수 있는 다중 분류를 위해 y_train.shape[1]의 값을 출력의 개수로 하고 활성화함수는 softmax를 이용하는 LSTM 모델을 구성하기로 했다. 그리고 데이터셋의 개수가 충분하지 않지만 훈련셋과 검증셋 시험셋 3가지로 데이터셋을 가지고 인공지능 모델을 학습시켜 학습과정을 살펴보고 과정이 좋은 모델을 선정해 평가하는 방식으로 실험을 진행하려고 한다.

하지만 현재 불러온 데이터는 Class열의 데이터셋이 숫자가 아닌 문자열로 되어 있기 때문에 이 문자열을 숫자로 변환시키는 replace함수를 이용했다. Baseline은 0 VT는 1 VF는 2로 변환하는 과정을 거쳐서 잘 변환이 되었는지 value_count().plot을 이용해 확인을 하고 잘 변환이 된 것을 알 수 있다.

```
df['Class'] = df['Class'].replace(['Baseline', 'VT', 'VF'],[0,1,2])
# Baseline은 0 VT는 1 VF는 2로 변환해서 코딩.
df['Class'].value_counts().plot(kind='bar')
#데이터의 분포가 Baseline, VT, VF의 비율이 일정하지 않음
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff88585d240>
```



아래의 코드처럼 매번 실행 시마다 동일 모델인데도 불구하고 다른 결과가 나오기 때문에 결과가 달라지지 않도록 np.random.seed(5)를 이용해 랜덤 시드를 명시적으로 지정을 하고 판다스로 읽어온 데이터 프레임을 넘파이 배열로 이용하기 위해 판다스의 데이터 프레임을 넘파이의 배열로 변환을 하는 코드인 mah_np_array = df.values를 이용해 dataset이라는 변수에 변환된 넘파이 배열을 저장했다. 넘파이 배열이 알맞게 출력이 되는 것을 확인할 수 있다. 이렇게 인공지능 모델을 구성하기 전 그리고 데이터셋을 훈련셋과 검증셋, 시험셋으로 생성하기 전인 데이터 준비과정이 끝났다.

```
# 랜덤시드 고정시키기
np.random.seed(5)
# 1 데이터 준비하기
mah_np_array = df.values
dataset = mah_np_array
print(dataset)
```

```
[[   3  970  970 ...  950  950    0]
 [   3  730  760 ...  240  240    1]
 [   3  620  940 ...  700  700    0]
 ...
 [8079  820  490 ...  270  280    1]
 [8096 1080 1070 ... 1200 1170    0]
 [8096  740  740 ...  320  290    1]]
```

3. Model development

이제 인공지능의 모델을 구성하려고 한다. 처음에 986개의 심전도 데이터를 입력으로 하는 입력층을 만들고 활성화 함수는 일반적으로 성능이 좋다고 알려진 디폴트값인 탄젠트 하이퍼볼릭 함수를 사용했다. 은닉층의 경우 2개의 층으로 만들고 마지막 출력층은 다중 분류를 위해 출력의 개수를 3으로 설정하고 활성화함수는 softmax함수를 사용해 모델을 구성했다.

```
#3. 모델 구성하기
model = Sequential()
#LSTM의 활성화함수 디폴트: 탄젠트 하이퍼볼릭 함수, relu를 써도 되지만 결과가 좋지 않음
model.add( LSTM(32, input_shape=(986,1)))
model.add( Dense(32,activation='relu'))
model.add( Dense(8,activation='relu'))
model.add(Dense(y_train.shape[1], activation= 'softmax'))
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 32)                4352
_________________________________________________________________
dense_9 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 27
=================================================================
Total params: 5,699
Trainable params: 5,699
Non-trainable params: 0
_________________________________________________________________
```

4. Dataset handling

이제 준비된 데이터셋을 훈련셋, 검증셋, 시험셋 3가지로 나누어서 구성을 하고 모델을 컴파일시키고 핏해서 학습을 시키고 학습과정을 지켜보고 괜찮은 학습 과정을 보인 모델을 이용해 시험셋을 평가 해보려고 한다.

기존 넘파이 배열에서 sample number 열과 Class 열을 제외한 심전도 986개의 데이터를 입력값으로 넣어주기 위해서 x의 데이터셋의 범위를 [:, 1:-1]로 설정했다. 심장의 상태를 나타내는 Class 열 1개의 출력값을 도출하기 위해서 y의 데이터셋의 범위를 [:,-1]로 설정했다. 그리고 다음과 같이 50개의 훈련셋 49개의 검증셋, 나머지를 시험셋으로 데이터셋을 생성했다.

```
#2 데이터셋 생성하기
# Training and testing dataset 분리, 필요시 validation dataset 분리
# Training dataset
x_train = dataset[:50,1:-1]
y_train = dataset[:50,-1]

# Validation dataset
x_val = dataset[50:100,1:-1]
y_val = dataset[50:100,-1]

# testing dataset
x_test = dataset[100:,1:-1]
y_test = dataset[100:,-1]
```

그리고 데이터셋을 LSTM 모델에 사용하기 위해 reshape함수를 이용해서 shape 형태를 조정해서 다시 각각의 훈련셋, 검증셋, 시험셋에 저장을 했다.

```
# 모델 변경, shape을 다시 잡아줌
# reshape for lstm
x_train = np.reshape(x_train,(len(x_train), 986, 1))
x_val = np.reshape(x_val,(len(x_val), 986, 1))
x_test = np.reshape(x_test,(len(x_test), 986, 1))
```

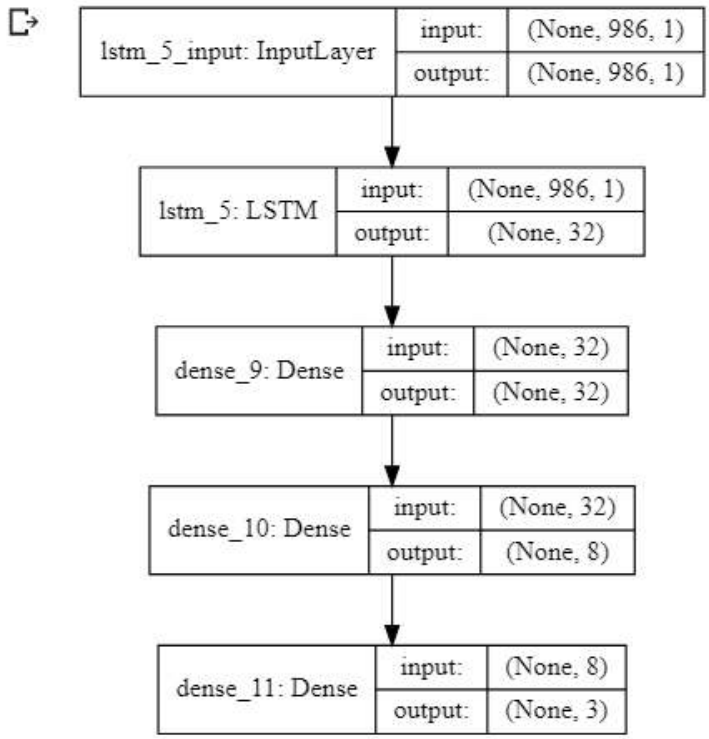
그리고 출력값을 라벨링 전환을 통해 변환시키고 다중 분류 모델이기 때문에 loss function의 값을 categorical_crossentropy로 학습 과정을 설정했다. 앞에서 만든 모델이 잘 구성되었는지 모델 시각화를 통해 그림을 나타냈다.

```
# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

```
#4 모델컴파일 학습과정 설정하기
# loss 현재 가중치 세트 평가하는데 사용한 손실 함수.
# optimizer 최적의 가중치 검색하는데 사용되는 최적화 알고리즘 효율적인 경사
 하강법 알고리즘 중 하나인 'adam'을 사용
# metrics 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로
지정

model.compile(loss='categorical_crossentropy', optimizer='adam', metri
cs=['accuracy'])

# 모델 시각화
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
%matplotlib inline
SVG(model_to_dot(model, show_shapes=True, dpi = 60).create(prog='dot',
 format='svg'))
```



fit함수를 이용해 학습을 시키는데 EarlyStopping을 호출해 적절한 조기종료를 시키게 코드를 구성하였고 epoch와 batcy_size를 조정하고 검증셋을 이용해 학습과정을 그래프로 볼수 있게 matplotlib함수를 사용하고 evaluate 함수를 이용해 평가를 하는 코드를 구성했다. 속도의 이점 때문에 배치 사이즈를 100으로 하고 실험을 하려고 한다.

```
#5 모델 학습시키기
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience = 20) # 조기종료 콜백함수 정의
hist= model.fit(x_train, y_train, epochs=500, batch_size=100,validatio
n_data=(x_val, y_val),callbacks=[early_stopping])


# 5. 모델 학습 과정 표시하기
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(1,1))
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()
loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')
acc_ax.plot(hist.history['accuracy'], 'b', label='train accuracy')
acc_ax.plot(hist.history['val_accuracy'], 'g', label='val accuracy')
loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')
loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')
plt.show()


# 학습 과정 살펴보기
print('## training loss and acc ##')
print(hist.history['loss'])
print(hist.history['accuracy'])
print(hist.history['val_loss'])
print(hist.history['val_accuracy'])

#6모델 평가하기
scores = model.evaluate(x_test, y_test)
print("acc: %f" %(scores[1]*100))
print('')
print('loss : ' + str(scores[0]))
print('accuray : ' + str(scores[1]))
```

5. Experiment settings & Performance (1)

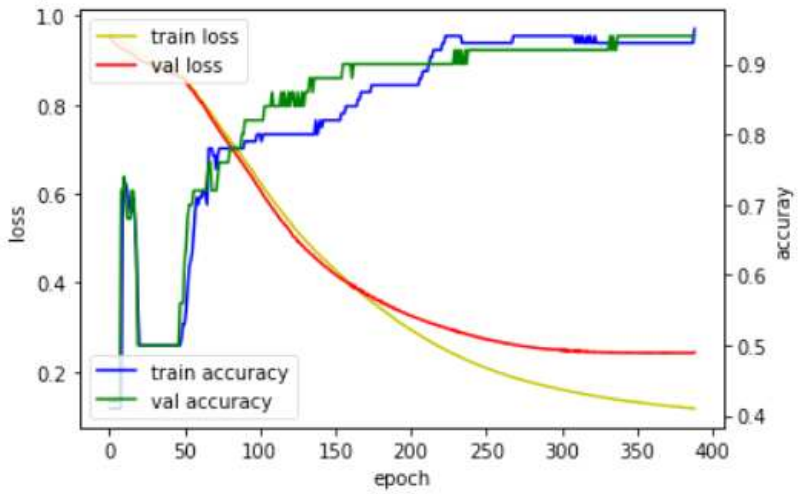
훈련셋 100개, 검증셋 50개, 시험셋: 나머지

```
#2 데이터셋 생성하기
# Training and testing dataset 분리, 필요시 validation dataset 분리
# Training dataset
x_train = dataset[0:100,1:-1]
# print(x_train)
y_train = dataset[0:100,-1]
print(y_train)
# print(y_train)
# Validation dataset
x_val = dataset[100:150,1:-1]
y_val = dataset[100:150,-1]
# print(x_val)
# testing dataset
x_test = dataset[150:,1:-1]
y_test = dataset[150:,-1]
# print(y_train, y_val, y_test)
# 모델 변경, shape을 다시 잡아줌
# reshape for lstm
x_train = np.reshape(x_train,(len(x_train), 986, 1))
x_val = np.reshape(x_val,(len(x_val), 986, 1))
x_test = np.reshape(x_test,(len(x_test), 986, 1))


# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

(1) patience = 20, epochs = 500, batch_size = 100

<Figure size 72x72 with 0 Axes>

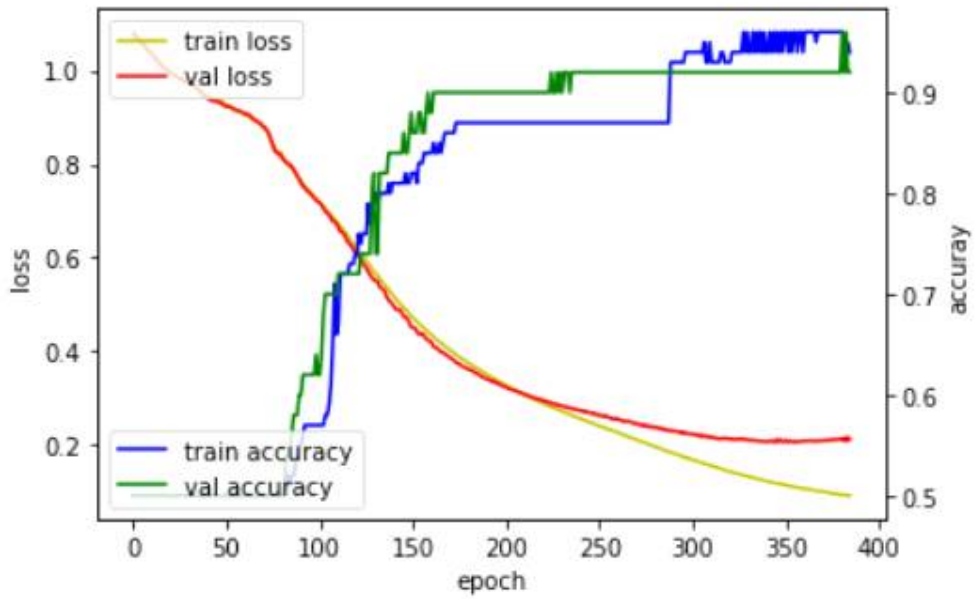



```
## training loss and acc ##
[0.9700406789779663, 0.9639754295349121, 0.9586326479911804, 0.954
[0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.72, 0.72,
[0.9543028473854065, 0.9488593935966492, 0.9436866044998169, 0.9397
[0.41999998688697815, 0.41999998688697815, 0.41999998688697815, 0.4
120/120 [==============================] - 0s 3ms/step
loss : 0.18178401986757914
accuray : 0.9250000011920929
acc: 92.500001
```

LSTM을 사용하니 batch_size를 100으로 크게 잡아도 batch_size를 작게 하고 적당한 patience를 줬었던 약 88의 정확도의 수치를 가진 DNN 모델보다 정확도가 높은 92.5인 훌륭한 결과가 나왔다.

(2) patience = 30, epochs = 1000, batch_size = 100

<Figure size 72x72 with 0 Axes>

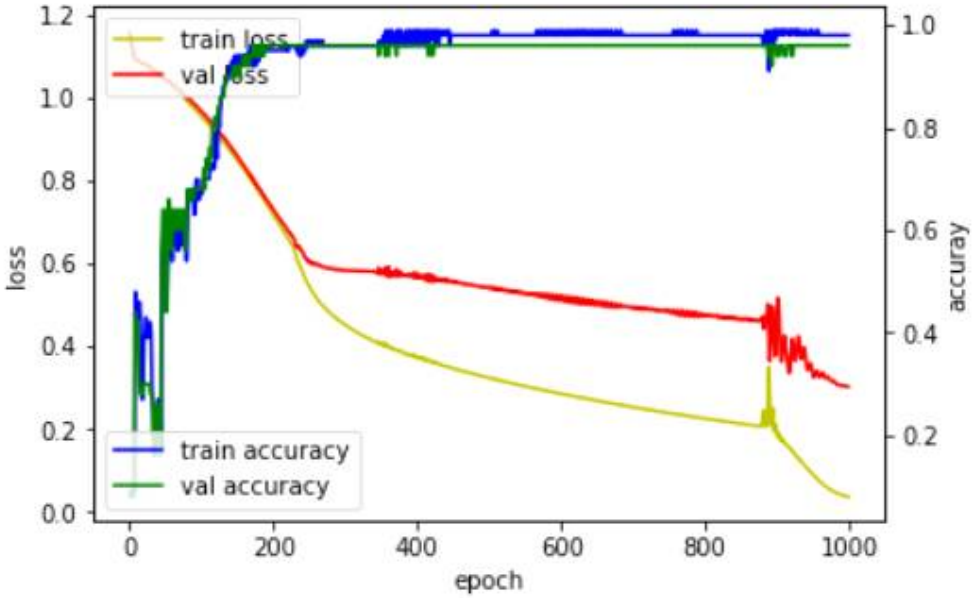



```
## training loss and acc ##
[1.0820395946502686, 1.0781161785125732, 1.0740362405776978, 1.06919062137603
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0
[1.0792171955108643, 1.0749921798706055, 1.0699383020401, 1.0652437210083008,
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0
120/120 [==============================] - 0s 2ms/step
loss : 0.17071203452845415
accuray : 0.9416666626930237
acc: 94.166666
```

386번째 epoch에서 조기 종료가 된 모습이고 시험셋에서 정확도가 대략 94.16이고 로스값은 0.17인 인공지능 모델이다. patience값을 30으로 증가시켰더니 정확도는 94.16으로 소폭 상승하였다.

(3) patience = 50, epochs = 1000, batch_size = 100

<Figure size 72x72 with 0 Axes>




```
## training loss and acc ##
[1.1647233963012695, 1.1531405448913574, 1.1443663835525513, 1.1345
[0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.48, 0.44, 0.43,
[1.1546685695648193, 1.1456509828567505, 1.1357548236846924, 1.1261
[0.07999999821186066, 0.07999999821186066, 0.07999999821186066, 0.0
120/120 [==============================] - 0s 3ms/step
loss : 0.1924502604951461
accuray : 0.9416666626930237
acc: 94.166666
```

1000번째 에포크까지 돌아간 모습이고 시험셋에서 정확도가 대략 94.16이고 로스값은 0.19인 인공지능 모델이다. patience값을 50으로 증가시켰지만 정확도는 94.16으로 비슷하였고 로스 값은 증가하였다.

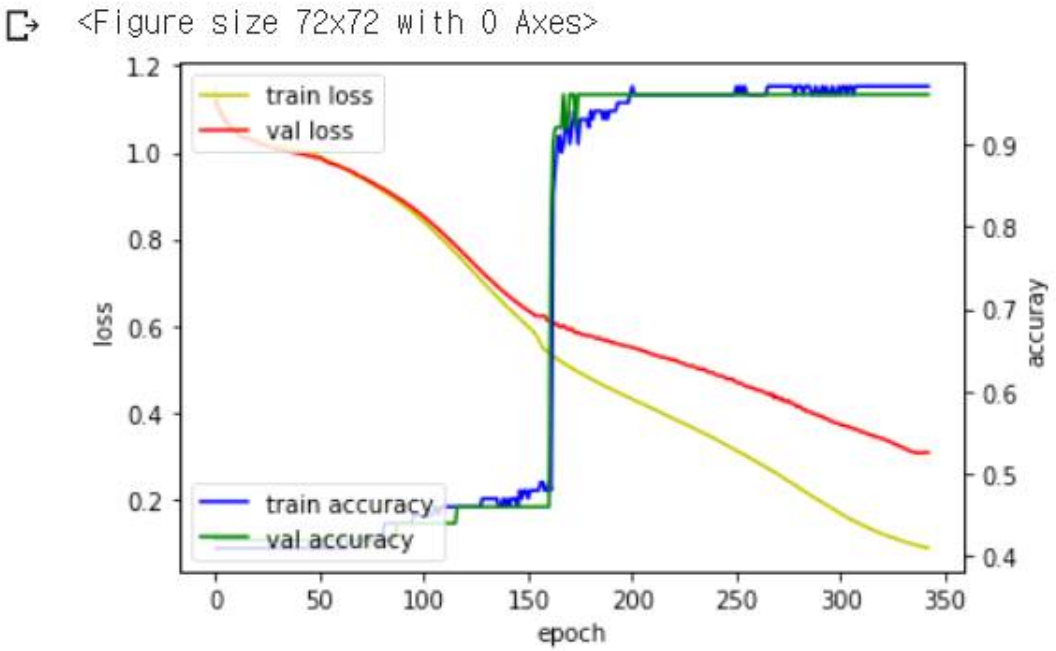
5. Experiment settings & Performance (2)

이번에는 데이터셋의 범위를 바꿔서 실험을 해보았다. 훈련셋은 150개로 증가시키고 검증셋 50개, 시험셋을 70개로 조정하였다.

```
#2 데이터셋 생성하기
# Training and testing dataset 분리, 필요시 validation dataset 분리
# Training dataset
x_train = dataset[0:150,1:-1]
# print(x_train)
y_train = dataset[0:150,-1]
print(y_train)
# print(y_train)
# Validation dataset
x_val = dataset[150:200,1:-1]
y_val = dataset[150:200,-1]
# print(x_val)
# testing dataset
x_test = dataset[200:,1:-1]
y_test = dataset[200:,-1]
# print(y_train, y_val, y_test)
# 모델 변경, shape을 다시 잡아줌
# reshape for lstm
x_train = np.reshape(x_train,(len(x_train), 986, 1))
x_val = np.reshape(x_val,(len(x_val), 986, 1))
x_test = np.reshape(x_test,(len(x_test), 986, 1))


# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

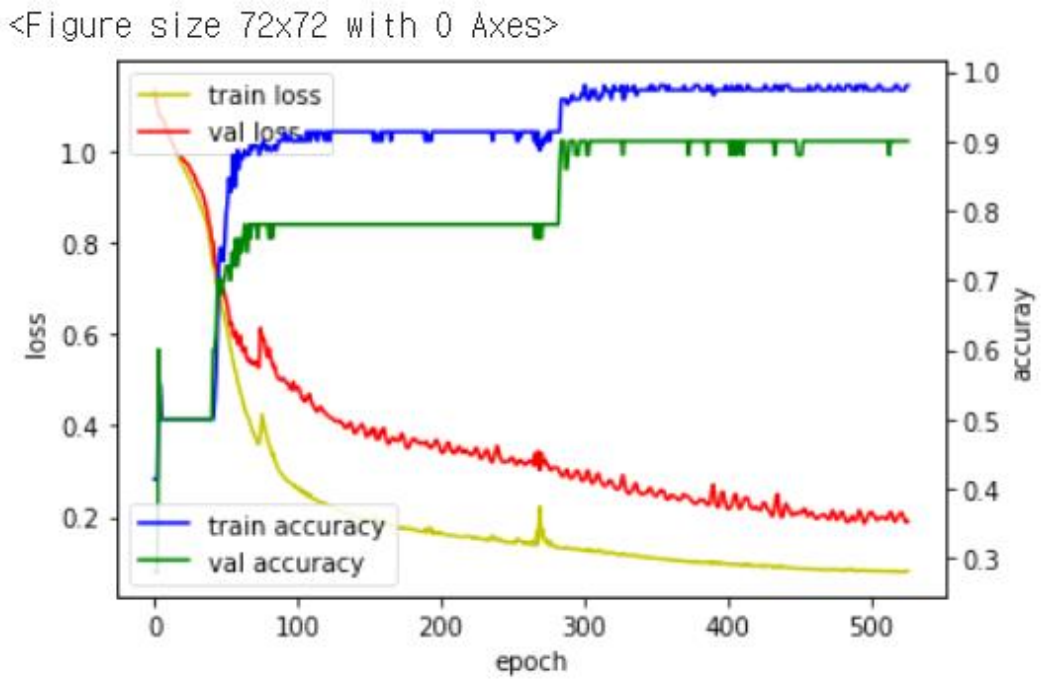
(1) patience = 20, epochs = 1000, batch_size = 100




```
## training loss and acc ##
[0.9570154547691345, 0.9500386516253153, 0.9418359200159708,
[0.41333333, 0.41333333, 0.41333333, 0.41333333, 0.41333333,
[1.12638258934021, 1.1193671226501465, 1.113747477531433, 1.
[0.2800000011920929, 0.2800000011920929, 0.2800000011920929,
70/70 [==============================] - 0s 3ms/step
loss : 0.13692740542548043
accuray : 0.9571428298950195
acc: 95.714283
```

347번째 epoch에서 조기 종료가 된 모습이고 시험셋에서 정확도가 대략 95.71이고 로스값은 0.13인 인공지능 모델이다. 훈련셋을 증가시키니 patience가 20일 때 정확도가 소폭 증가되었다.

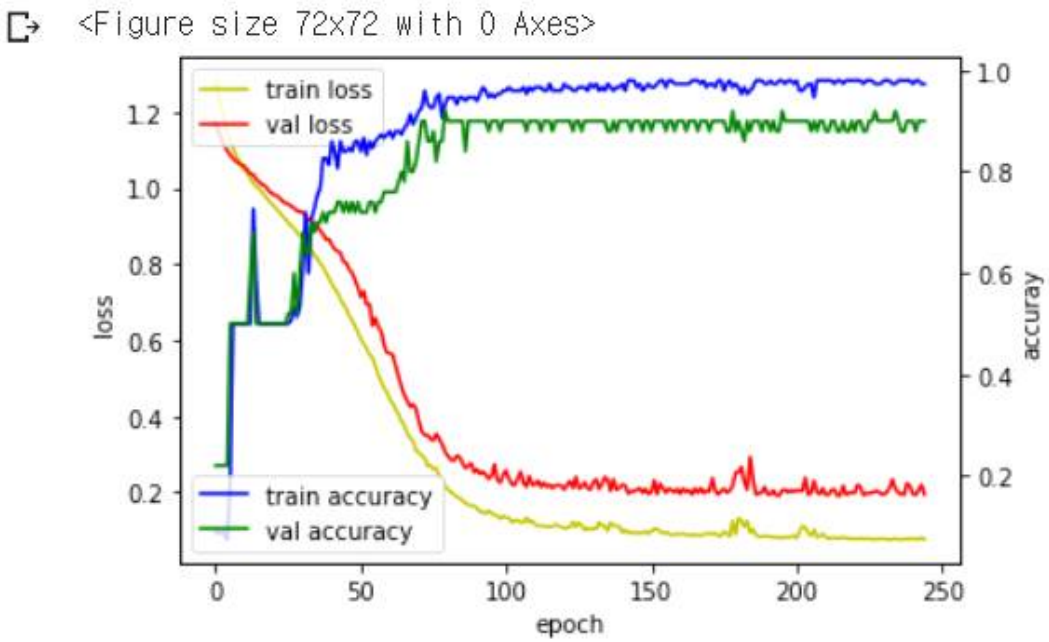
(2) patience = 30, epochs = 1000, batch_size = 100




```
## training loss and acc ##
[1.1432470083236694, 1.1185576518376668, 1.1017400821050007,
[0.41333333, 0.41333333, 0.41333333, 0.56, 0.5466667, 0.5, 0
[1.1301007270812988, 1.1110886335372925, 1.0950708389282227,
[0.2800000011920929, 0.2800000011920929, 0.2800000011920929,
70/70 [==============================] - 0s 3ms/step
loss : 0.050674668167318616
accuray : 0.9857142567634583
acc: 98.571426
```

527번째 epoch에서 조기 종료가 된 모습이고 시험셋에서 정확도가 대략 98.57이고 로스값은 0.05인 인공지능 모델이다. patience값을 올려줬더니 훨씬 더 좋은 모델이 되었다. 속도의 이점 때문에 배치사이즈 값을 100으로 설정했었다.
배치 사이즈 값이 100인데도 훌륭한 정확도를 가진 모델이 나왔기에 배치 사이즈를 50으로 줄여서 학습을 시켜보려고 한다.

(3) patience = 30, epochs = 1000, batch_size = 50

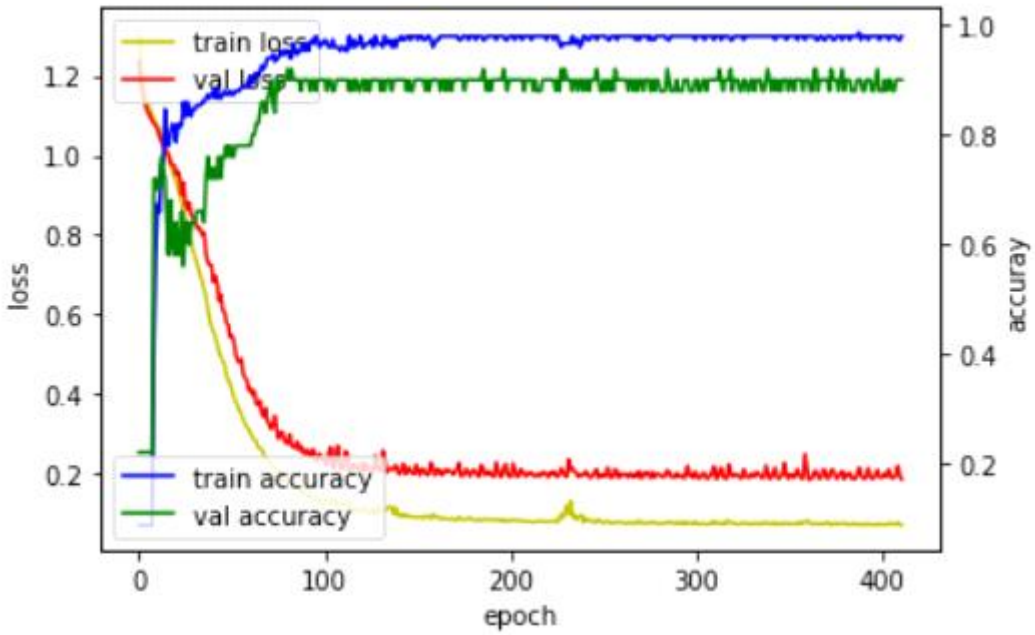



```
## training loss and acc ##
[1.2834835449854534, 1.2449339628219604, 1.2107208172
[0.086666666, 0.086666666, 0.086666666, 0.086666666,
[1.1749167442321777, 1.149329662322998, 1.12923395633
[0.2199999988079071, 0.2199999988079071, 0.219999998
70/70 [==============================] - 0s 3ms/step
loss : 0.05323289803096226
accuray : 0.9857142567634583
acc: 98.571426
```

245번째 epoch에서 조기 종료가 된 모습이고 시험셋에서 정확도가 대략 98.57이고 로스값은 0.05인 인공지능 모델이다. 배치 사이즈값을 작게 해줬더니 좀 더 이른 epoch에서 학습이 조기 종료 되었고 성능이 더 좋아진 모델이 되었다.

(4) patience = 50, epochs = 1000, batch_size = 50

<Figure size 72x72 with 0 Axes>




```
## training loss and acc ##
[1.3086785078048706, 1.246267318725586, 1.197039405504
[0.086666666, 0.086666666, 0.086666666, 0.086666666, C
[1.2380973100662231, 1.180391788482666, 1.152349829673
[0.2199999988079071, 0.2199999988079071, 0.2199999988C
70/70 [==============================] - 0s 3ms/step
loss : 0.04792827538081578
accuray : 0.9857142567634583
acc: 98.571426
```

411번째 epoch에서 조기 종료가 된 모습이고 시험셋에서 정확도가 대략 98.57이고 로스값은 0.047인 인공지능 모델이다. patience값을 50으로 증가시켰지만 기존 patience값이 30일 때와 큰 차이가 없었다.

5. Experiment settings & Performance (3)

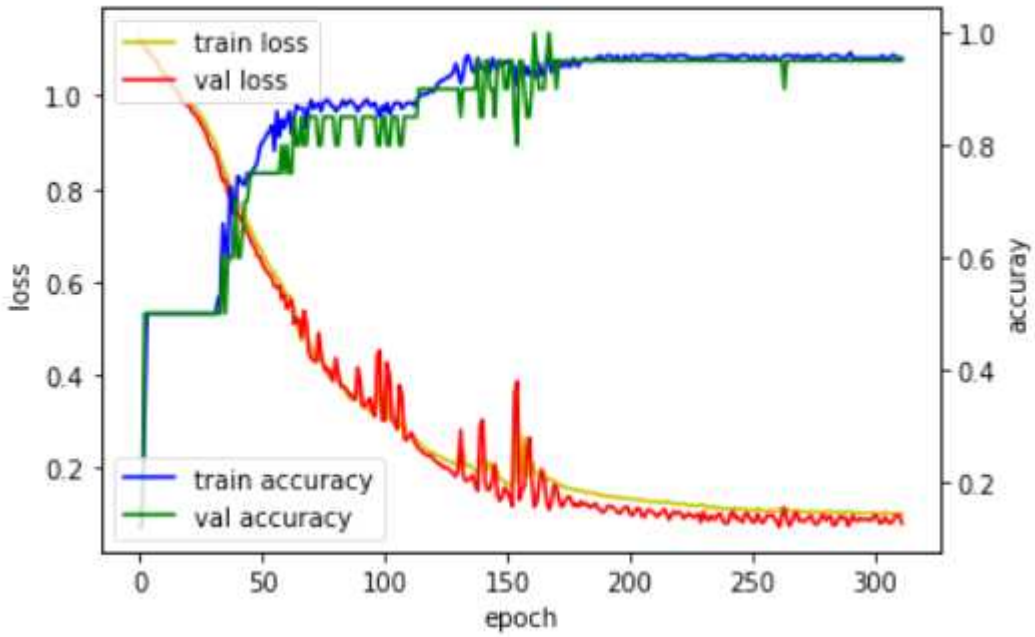
데이터셋의 범위를 이번에는 훈련셋을 200개, 검증셋을 20개, 나머지를 평가셋으로 설정하고 실험을 진행했다. patient값은 50으로 고정시키고 batch_size값에 따라 실험을 진행하려고 한다.

```
#2 데이터셋 생성하기
# Training and testing dataset 분리, 필요시 validation dataset 분리
# Training dataset
x_train = dataset[0:200,1:-1]
# print(x_train)
y_train = dataset[0:200,-1]
print(y_train)
# print(y_train)
# Validation dataset
x_val = dataset[200:220,1:-1]
y_val = dataset[200:220,-1]
# print(x_val)
# testing dataset
x_test = dataset[220:,1:-1]
y_test = dataset[220:,-1]
# print(y_train, y_val, y_test)
# 모델 변경, shape을 다시 잡아줌
# reshape for lstm
x_train = np.reshape(x_train,(len(x_train), 986, 1))
x_val = np.reshape(x_val,(len(x_val), 986, 1))
x_test = np.reshape(x_test,(len(x_test), 986, 1))


# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

(1) patience = 50, epochs = 1000, batch_size = 100

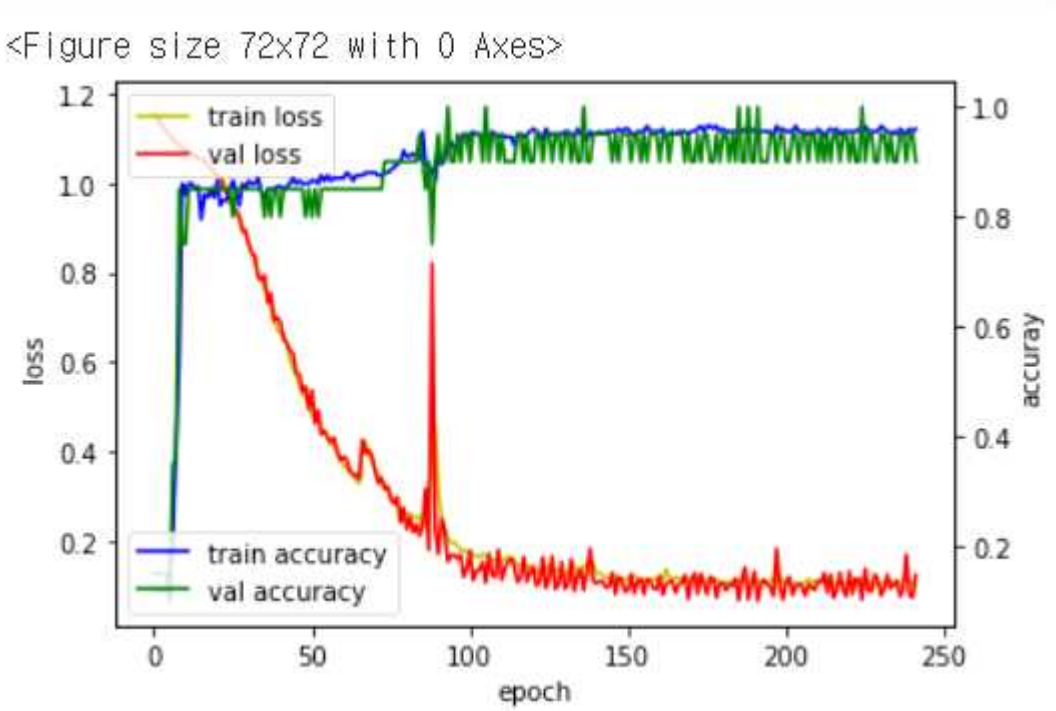
<Figure size 72x72 with 0 Axes>




```
## training loss and acc ##
[1.1355028748512268, 1.1263012886047363, 1.1174476742
[0.12, 0.12, 0.32, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
[1.1198612451553345, 1.1116304397583008, 1.1039454936
[0.15000000596046448, 0.15000000596046448, 0.5, 0.5,
50/50 [==============================] - 0s 3ms/step
loss : 0.04206164702773094
accuray : 1.0
acc: 100.000000
```

312번째 epoch에서 조기 종료가 된 모습이고 학습 데이터의 수가 50개가 더 많아지니 정확도가 100으로 이전 모델보다 정확도가 높아지고 loss 값이 0.042이다. 평가 데이터는 다 맞추는 좋은 모델이 되었다.

(2) patience = 50, epochs = 1000, batch_size = 50




```
## training loss and acc ##
[1.1704885065555573, 1.1530753076076508, 1.14047816395
[0.12, 0.12, 0.12, 0.12, 0.12, 0.11, 0.22, 0.38, 0.535
[1.1529343128204346, 1.1422779560089111, 1.13288700580
[0.15000000596046448, 0.15000000596046448, 0.150000005
50/50 [==============================] - 0s 3ms/step
loss : 0.024347277805209 16
accuray : 1.0
acc: 100.000000
```

242번째 epoch에서 조기 종료가 된 모습이고 정확도가 100으로 이전 모델과 같이 정확도가 100인 인공지능 모델이 되었다. loss값이 0.024로 소폭 떨어진 것을 알 수 있고 더 적은 epoch로 좋은 성능을 내는 모델이 되었다.

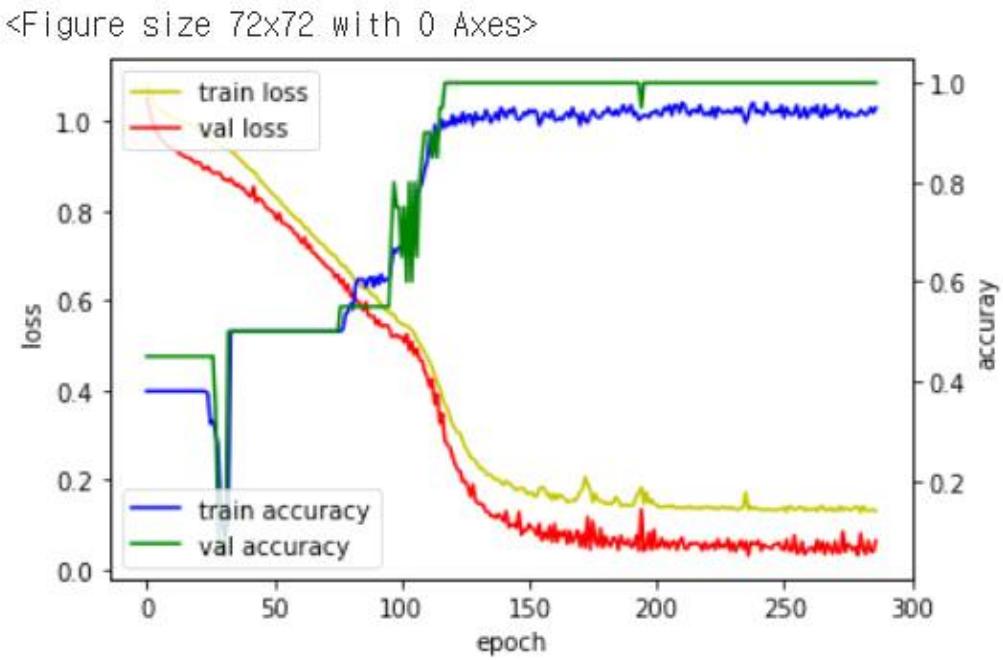
5. Experiment settings & Performance (5)

데이터셋의 범위를 이번에는 훈련셋 200개는 그대로고 검증셋을 20개, 나머지를 평가셋으로 설정하고 실험을 진행했다. 하지만 위의 실험과 다르게 이번에는 검증셋과 평가셋의 범위를 약간 바꿔서 시행할 생각이다. patient값은 이번에도 50으로 고정시키고 batch_size값에 따라 실험을 진행하려고 한다.

```
#2 데이터셋 생성하기
# Training and testing dataset 분리, 필요시 validation dataset 분리
# Training dataset
x_train = dataset[0:200,1:-1]
# print(x_train)
y_train = dataset[0:200,-1]
print(y_train)
# print(y_train)
# Validation dataset
x_val = dataset[250:,1:-1]
y_val = dataset[250:,-1]
# print(x_val)
# testing dataset
x_test = dataset[200:250,1:-1]
y_test = dataset[200:250:,-1]
# print(y_train, y_val, y_test)
# 모델 변경, shape을 다시 잡아줌
# reshape for lstm
x_train = np.reshape(x_train,(len(x_train), 986, 1))
x_val = np.reshape(x_val,(len(x_val), 986, 1))
x_test = np.reshape(x_test,(len(x_test), 986, 1))


# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

(1) patience = 50, epochs = 1000, batch_size = 100




```
## training loss and acc ##
[1.0848892629146576, 1.0651006996631622, 1.0518133938
[0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38,
[1.0510404109954834, 1.0265233516693115, 1.0047566890
[0.44999998807907104, 0.44999998807907104, 0.4499999
50/50 [==============================] - 0s 3ms/step
loss : 0.059798314422369006
accuray : 1.0
acc: 100.000000
```

274번째 epoch에서 조기 종료가 된 모습이고 학습 데이터의 수가 50개가 더 많아지니 정확도가 100으로 이전 모델보다 정확도가 높아지고 loss 값이 0.042이다. 평가 데이터는 다 맞추는 좋은 모델이 되었다.