ANN

Keras Classification Code Along

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

In [2]:
```
# df = pd.read_csv(r"D:\LIUZHICHENG\Udemy\Deep Learning\Jose\TF_2_Notebooks_and_Data\DATA\cancer_classification.csv")
df = pd.read_csv("/Volumes/GoogleDrive/My Drive/Udemy/Deep Learning/Jose/TF_2_Notebooks_and_Data/DATA/cancer_classificat
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	c
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	

5 rows × 31 columns

In [3]:
```
df.shape
```

Out[3]: (569, 31)

In [4]:
```
df.isna().sum()
```

Out[4]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
benign_0__mal_1            0
dtype: int64
```

EDA for new df

[...]

Building a Model

In [7]:
```
X = df.drop(columns="benign_0__mal_1").to_numpy()
y = df["benign_0__mal_1"].to_numpy()
```

In [8]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101)

from sklearn.preprocessing import MinMaxScaler, StandardScaler

sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(layer=Dense(units=30, activation="relu"))
model.add(layer=Dense(units=15, activation="relu"))

# Binary classification
model.add(layer=Dense(units=1, activation="sigmoid"))  # last layer

model.compile(optimizer="adam", loss="binary_crossentropy")
```

In [10]:
```
model.fit(x=X_train, y=y_train, epochs=600,
          validation_data=(X_test, y_test),
          verbose=1)
```

```
Epoch 1/600
14/14 [==============================] - 1s 10ms/step - loss: 0.6611 - val_loss: 0.6341
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6171 - val_loss: 0.5904
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5717 - val_loss: 0.5439
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5250 - val_loss: 0.4943
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4745 - val_loss: 0.4428
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4295 - val_loss: 0.3957
Epoch 7/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3842 - val_loss: 0.3532
Epoch 8/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3470 - val_loss: 0.3163
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3129 - val_loss: 0.2832
Epoch 10/600
14/14 [==============================] - 0s 2ms/step - loss: 0.2886 - val_loss: 0.2613
```

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                930
_________________________________________________________________
dense_1 (Dense)              (None, 15)                465
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 16
=================================================================
Total params: 1,411
Trainable params: 1,411
Non-trainable params: 0
_________________________________________________________________
```

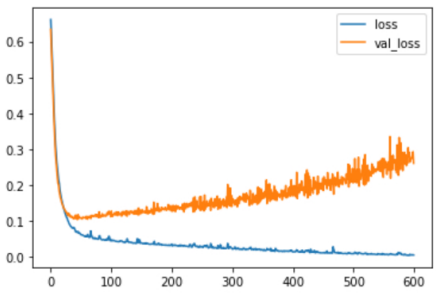
In [12]:
```
loss_df = pd.DataFrame(data=model.history.history)
loss_df
# loss, validation loss
```

Out[12]:

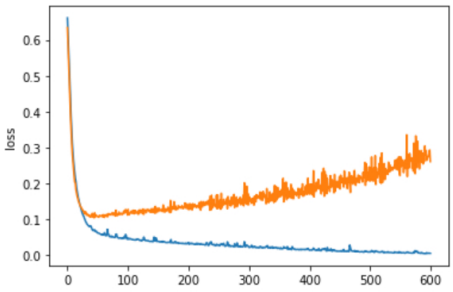
	loss	val_loss
0	0.661094	0.634076
1	0.617055	0.590358
2	0.571726	0.543933
3	0.525043	0.494313
4	0.474459	0.442837
...	...	...
595	0.005870	0.272632
596	0.006248	0.278575
597	0.006599	0.283509
598	0.006501	0.293628
599	0.006030	0.261371

600 rows × 2 columns

In [13]:
```
# overfitting
loss_df.plot();
```



In [14]:
```
sns.lineplot(data=loss_df, x=loss_df.index, y="loss")
sns.lineplot(data=loss_df, x=loss_df.index, y="val_loss");
```



EarlyStopping

In [15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(layer=Dense(units=30, activation="relu"))
model.add(layer=Dense(units=15, activation="relu"))

# Binary classification
model.add(layer=Dense(units=1, activation="sigmoid"))  # last layer

model.compile(optimizer="adam", loss="binary_crossentropy")

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor="val_loss", mode="min", verbose=1, patience=25)

model.fit(x=X_train, y=y_train, epochs=600,
          validation_data=(X_test, y_test),
          verbose=1, callbacks=[early_stop])
```

```
Epoch 1/600
14/14 [==============================] - 0s 8ms/step - loss: 0.6681 - val_loss: 0.6541
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6345 - val_loss: 0.6214
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5998 - val_loss: 0.5871
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5647 - val_loss: 0.5477
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5232 - val_loss: 0.5060
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4811 - val_loss: 0.4590
Epoch 7/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4305 - val_loss: 0.4067
Epoch 8/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3823 - val_loss: 0.3600
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3415 - val_loss: 0.3193
Epoch 10/600
14/14 [==============================] - 0s 2ms/step - loss: 0.3062 - val_loss: 0.2860
```

In [16]:
```
model.history.history
```

Out[16]:
```
{'loss': [0.6681123971939087,
  0.6344802975654602,
  0.5997874140739441,
  0.564723014831543,
  0.5232142210006714,
  0.48110640048980713,
  0.4304735064506531,
  0.38230839371681213,
  0.34154751896858215,
  0.306238055229187,
  0.2775701880455017,
  0.25517910718917847,
  0.2378966063261032,
  0.22003865242004395,
  0.2091105580329895,
  0.19965435564517975,
  0.18735867738723755,
  0.17597214877605438,
  0.16939526796340942,
  0.1586430370807647,
```

In [17]:
```
loss_df = pd.DataFrame(data=model.history.history)
loss_df
```

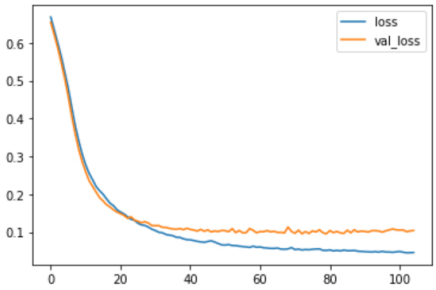
Out[17]:

	loss	val_loss
0	0.668112	0.654135
1	0.634480	0.621362
2	0.599787	0.587133
3	0.564723	0.547748
4	0.523214	0.506039
...	...	...

100	0.048395	0.105124
101	0.046471	0.105373
102	0.045163	0.100747
103	0.045341	0.102875
104	0.045771	0.103905

105 rows × 2 columns

In [18]:
```
loss_df.plot();
```



Dropout

In [19]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(layer=Dense(units=30, activation="relu"))
model.add(layer=Dropout(rate=0.5)) # 50% neurons be turned off

model.add(layer=Dense(units=15, activation="relu"))
model.add(layer=Dropout(rate=0.5)) # 50% neurons be turned off

# Binary classification
model.add(layer=Dense(units=1, activation="sigmoid"))  # last layer

model.compile(optimizer="adam", loss="binary_crossentropy")

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor="val_loss", mode="min", verbose=1, patience=25)

model.fit(x=X_train, y=y_train, epochs=600,
          validation_data=(X_test, y_test),
          verbose=1, callbacks=[early_stop])
```

```
Epoch 1/600
14/14 [==============================] - 0s 8ms/step - loss: 0.6976 - val_loss: 0.6665
Epoch 2/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6719 - val_loss: 0.6579
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6706 - val_loss: 0.6471
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6563 - val_loss: 0.6329
Epoch 5/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6432 - val_loss: 0.6101
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6226 - val_loss: 0.5866
Epoch 7/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5918 - val_loss: 0.5583
Epoch 8/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5893 - val_loss: 0.5337
Epoch 9/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5681 - val_loss: 0.5093
Epoch 10/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5608 - val_loss: 0.4897
```

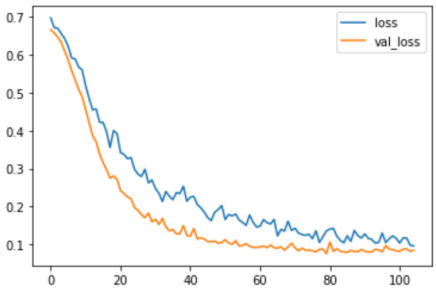
In [20]:
```
loss_df = pd.DataFrame(data=model.history.history)
loss_df
```

Out[20]:

	loss	val_loss
0	0.697633	0.666492
1	0.671942	0.657884
2	0.670568	0.647124
3	0.656304	0.632946
4	0.643187	0.610075
...	...	...
100	0.103479	0.082177
101	0.117253	0.087561
102	0.117101	0.088553
103	0.098061	0.081630
104	0.096125	0.084610

105 rows × 2 columns

In [21]:
```
loss_df.plot();
```



model.predict_classes instead of model.predict

In [22]:
```
model.predict_classes(X_test)
```

```
/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

Out[22]:
```
array([[1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [0],
       [1],
       [1],
```

In [23]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(classification_report(y_test, model.predict_classes(X_test)))
print(confusion_matrix(y_test, model.predict_classes(X_test)))
print(accuracy_score(y_test, model.predict_classes(X_test)))
```

```
/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        55
           1       0.99      0.98      0.98        88

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143

[[54  1]
 [ 2 86]]
0.9790209790209791
```