Chapter 3: ANN

Iris 2 outputs

- Cho dữ liệu Iris.xls.
- Xây dựng model để dự đoán: petallength và loại iris từ những thuộc tính còn lại.

In [1]:
```
import pandas as pd
from sklearn.datasets import load_iris
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input, Model
import tensorflow as tf
```

In [2]:
```
data = pd.read_excel("Iris.xls")
```

In [3]:
```
data
```

Out[3]:

	sepallength	sepalwidth	petallength	petalwidth	iris
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [4]:
```
iris_class = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2}
data['species_num'] = [iris_class[i] for i in data.iris]
data.head()
```

Out[4]:

	sepallength	sepalwidth	petallength	petalwidth	iris	species_num
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0

In [5]:
```
df_train = data.sample(frac=0.7, random_state=0)
df_valid = data.drop(df_train.index)
```

In [6]:
```
X_train = df_train.drop(['petallength', 'iris', 'species_num'], axis=1)
X_valid = df_valid.drop(['petallength', 'iris', 'species_num'], axis=1)
y_train = df_train['petallength']
y_valid = df_valid['petallength']
z_train = df_train['species_num']
z_valid = df_valid['species_num']
```

Build model

In [12]:
```
inputs = Input(shape=(3,), name='input')
x = Dense(16, activation='relu', name='16')(inputs)
x = Dense(32, activation='relu', name='32')(x)
output1 = Dense(1, name='cont_out')(x)
output2 = Dense(3, activation='softmax', name='cat_out')(x)
```

In [13]:
```
model = Model(inputs=inputs, outputs=[output1, output2])
```

In [14]:
```
model.compile(loss={'cont_out': 'mean_absolute_error',
                    'cat_out': 'sparse_categorical_crossentropy'},
              optimizer='adam',
              )
```

In [15]:
```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
_____________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
==================
input (InputLayer)              [(None, 3)]          0
__________________________________________________________________________________________
_____________________
16 (Dense)                      (None, 16)           64          input[0][0]
__________________________________________________________________________________________
_____________________
32 (Dense)                      (None, 32)           544         16[0][0]
__________________________________________________________________________________________
_____________________
cont_out (Dense)                (None, 1)            33          32[0][0]
__________________________________________________________________________________________
_____________________
cat_out (Dense)                 (None, 3)            99          32[0][0]
==========================================================================================
==================
Total params: 740
Trainable params: 740
Non-trainable params: 0
__________________________________________________________________________________________
_____________________
```

In [16]:
```
history = model.fit(X_train, {'cont_out': y_train, 'cat_out': z_train},
                    validation_data=(X_valid,
                                     {'cont_out': y_valid, 'cat_out': z_valid}),
                    epochs=100,
                    batch_size=32)
```

```
Epoch 1/100
4/4 [==============================] - 1s 59ms/step - loss: 4.4890 - cont_out
_loss: 3.3610 - cat_out_loss: 1.1280 - val_loss: 4.1535 - val_cont_out_loss:
3.0099 - val_cat_out_loss: 1.1436
Epoch 2/100
4/4 [==============================] - 0s 7ms/step - loss: 4.2589 - cont_out_
loss: 3.1443 - cat_out_loss: 1.1146 - val_loss: 3.9303 - val_cont_out_loss:
2.7945 - val_cat_out_loss: 1.1359
Epoch 3/100
4/4 [==============================] - 0s 7ms/step - loss: 4.0304 - cont_out_
loss: 2.9303 - cat_out_loss: 1.1001 - val_loss: 3.7071 - val_cont_out_loss:
2.5783 - val_cat_out_loss: 1.1288
Epoch 4/100
4/4 [==============================] - 0s 7ms/step - loss: 3.8044 - cont_out_
loss: 2.7150 - cat_out_loss: 1.0894 - val_loss: 3.4854 - val_cont_out_loss:
2.3622 - val_cat_out_loss: 1.1232
Epoch 5/100
4/4 [==============================] - 0s 7ms/step - loss: 3.5870 - cont_out_
loss: 2.5063 - cat_out_loss: 1.0807 - val_loss: 3.2764 - val_cont_out_loss:
```

In [17]:
```
history_df = pd.DataFrame(history.history)
history_df
```

Out[17]:

	loss	cont_out_loss	cat_out_loss	val_loss	val_cont_out_loss	val_cat_out_loss
0	4.489001	3.360959	1.128042	4.153503	3.009895	1.143608
1	4.258904	3.144342	1.114562	3.930336	2.794450	1.135886
2	4.030402	2.930259	1.100144	3.707051	2.578289	1.128762
3	3.804382	2.714979	1.089403	3.485424	2.362179	1.123246
4	3.586967	2.506258	1.080709	3.276372	2.156590	1.119782
...	...	...	...	...	...	...
95	0.700567	0.251115	0.449452	0.716027	0.290470	0.425557
96	0.693234	0.248688	0.444546	0.697762	0.278180	0.419582
97	0.685174	0.245842	0.439332	0.704523	0.287018	0.417505
98	0.678408	0.243723	0.434685	0.707225	0.291880	0.415345
99	0.680372	0.248832	0.431540	0.703991	0.290933	0.413058

100 rows × 6 columns

In [18]:
```
history_df.loc[:, ['loss', 'cont_out_loss', 'cat_out_loss']].plot()
print("Minimum loss: {}".format(history_df['loss'].min()))
```

```
Minimum loss: 0.6784082055091858
```

In [19]:
```
y_z_hat_valid = model.predict(X_valid)
```

In [20]:
```
y_z_hat_valid[0][0:5]
```

Out[20]:
```
array([[1.4705153],
       [1.5446855],
       [1.6561188],
       [1.3450912],
       [1.5139712]], dtype=float32)
```

In [21]:
```
y_z_hat_valid[1][0:5]
```

Out[21]:
```
array([[0.9320144 , 0.05727227, 0.01071325],
       [0.8696561 , 0.10754735, 0.02279657],
       [0.9532271 , 0.04063237, 0.00614048],
       [0.9624346 , 0.03227636, 0.00528897],
       [0.9459632 , 0.04537204, 0.00866475]], dtype=float32)
```

In [34]:
```
y_z_hat_valid[1][-5:]
```

Out[34]:
```
array([[0.01911662, 0.28818014, 0.6927032 ],
       [0.02964595, 0.38770163, 0.58265233],
       [0.00542115, 0.24090001, 0.7536788 ],
       [0.01607166, 0.3245532 , 0.6593752 ],
       [0.00514989, 0.25002003, 0.74483   ]], dtype=float32)
```

In [29]:
```
import numpy as np
```

In [30]:
```
# tìm index của giá trị p lớn nhất của mỗi mẫu => loại hoa
index_array = np.argmax(y_z_hat_valid[1], axis=-1)
```

In [31]:
```
index_array[:5]
```

Out[31]:
```
array([0, 0, 0, 0, 0], dtype=int64)
```

In [33]:
```
index_array[-5:]
```

Out[33]:
```
array([2, 2, 2, 2, 2], dtype=int64)
```

In [27]:
```
y_valid[:5]
```

Out[27]:
```
0     1.4
9     1.5
14    1.2
19    1.5
21    1.5
Name: petallength, dtype: float64
```

In [26]:
```
z_valid[:5]
```

Out[26]:
```
0     0
9     0
14    0
19    0
21    0
Name: species_num, dtype: int64
```

In [35]:
```
z_valid[-5:]
```

Out[35]:
```
136    2
138    2
140    2
142    2
145    2
Name: species_num, dtype: int64
```

In [ ]:
```

```