Predicting flowers' variety IRIS datset with Deep learning Model

1- Example 1: Numerical IRIS dataset : without categorical variables

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.datasets import load_iris
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target
data
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
```

```
            [5.2, 3.5, 1.5, 0.2],
            [5.2, 3.4, 1.4, 0.2],
            [4.7, 3.2, 1.6, 0.2],
            [4.8, 3.1, 1.6, 0.2],
            [5.4, 3.4, 1.5, 0.4],
            [5.2, 4.1, 1.5, 0.1],
            [5.5, 4.2, 1.4, 0.2],
            [4.9, 3.1, 1.5, 0.2],
            [5. , 3.2, 1.2, 0.2],
            [5.5, 3.5, 1.3, 0.2],
            [4.9, 3.6, 1.4, 0.1],
            [4.4, 3. , 1.3, 0.2],
            [5.1, 3.4, 1.5, 0.2],
            [5. , 3.5, 1.3, 0.3],
            [4.5, 2.3, 1.3, 0.3],
            [4.4, 3.2, 1.3, 0.2],
            [5. , 3.5, 1.6, 0.6],
            [5.1, 3.8, 1.9, 0.4],
            [4.8, 3. , 1.4, 0.3],
            [5.1, 3.8, 1.6, 0.2],
            [4.6, 3.2, 1.4, 0.2],
            [5.3, 3.7, 1.5, 0.2],
            [5. , 3.3, 1.4, 0.2],
            [7. , 3.2, 4.7, 1.4],
            [6.4, 3.2, 4.5, 1.5],
            [6.9, 3.1, 4.9, 1.5],
            [5.5, 2.3, 4. , 1.3],
            [6.5, 2.8, 4.6, 1.5],
            [5.7, 2.8, 4.5, 1.3],
            [6.3, 3.3, 4.7, 1.6],
            [4.9, 2.4, 3.3, 1. ],

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Preprocess the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and compile the model
model = Sequential()
model.add(Dense(12, input_dim=4, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))


model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 12)                60
```

```
 dense_7 (Dense)             (None, 8)                 104

 dense_8 (Dense)             (None, 3)                 27

=================================================================
Total params: 191 (764.00 Byte)
Trainable params: 191 (764.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=1)

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/50
24/24 [==============================] - 1s 2ms/step - loss: 0.9749 - accur
Epoch 2/50
24/24 [==============================] - 0s 2ms/step - loss: 0.8859 - accur
Epoch 3/50
24/24 [==============================] - 0s 2ms/step - loss: 0.8177 - accur
Epoch 4/50
24/24 [==============================] - 0s 2ms/step - loss: 0.7516 - accur
Epoch 5/50
24/24 [==============================] - 0s 2ms/step - loss: 0.6841 - accur
Epoch 6/50
24/24 [==============================] - 0s 2ms/step - loss: 0.6174 - accur
Epoch 7/50
24/24 [==============================] - 0s 2ms/step - loss: 0.5588 - accur
Epoch 8/50
24/24 [==============================] - 0s 2ms/step - loss: 0.5106 - accur
Epoch 9/50
24/24 [==============================] - 0s 2ms/step - loss: 0.4705 - accur
Epoch 10/50
24/24 [==============================] - 0s 2ms/step - loss: 0.4387 - accur
Epoch 11/50
24/24 [==============================] - 0s 2ms/step - loss: 0.4147 - accur
Epoch 12/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3925 - accur
Epoch 13/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3742 - accur
Epoch 14/50
24/24 [==============================] - 0s 3ms/step - loss: 0.3573 - accur
Epoch 15/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3407 - accur
Epoch 16/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3262 - accur
Epoch 17/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3129 - accur
Epoch 18/50
24/24 [==============================] - 0s 2ms/step - loss: 0.3001 - accur
Epoch 19/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2889 - accur
Epoch 20/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2767 - accur
```

```
24/24 [==============================] - 0s 2ms/step - loss: 0.2767 - accur
Epoch 21/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2640 - accur
Epoch 22/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2543 - accur
Epoch 23/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2417 - accur
Epoch 24/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2320 - accur
Epoch 25/50
24/24 [==============================] - 0s 2ms/step - loss: 0.2179 - accur
Epoch 26/50
24/24 [==============================] - 0s 12ms/step - loss: 0.2088 - accu
Epoch 27/50
24/24 [==============================] - 0s 2ms/step - loss: 0.1987 - accur
Epoch 28/50
24/24 [==============================] - 0s 2ms/step - loss: 0.1904 - accur
Epoch 29/50
24/24 [==============================] - 0s 3ms/step - loss: 0.1818 - accur
```

2- Example 2 : IRIS dataset with categorical variables

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense



 #Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)


# fix random seed for reproducibility
seed = 42
np.random.seed(seed)


dataset
```

```
# Split dataset
Y = dataset['class']
X = dataset.drop(['class'], axis=1)
print("Shape of Input  features: {}".format(X.shape))
print("Shape of Output features: {}".format(Y.shape))
```

```
Shape of Input  features: (150, 4)
Shape of Output features: (150,)
```

Encoding the Output Variable

```
Y.value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: class, dtype: int64
```

1. Use ***LabelEncoder*** when you have ordinal categorical variables* **with a clear order or ranking*** among the categories.
2. Use ***OneHotEncoder*** when you have nominal categorical variables ***without a clear order*** among the categories, and you want to avoid implying any ordinal relationship.

```
label_encoder = LabelEncoder()

# Fit and transform the target variable
Y_encoded = label_encoder.fit_transform(Y)

Y_final = tf.keras.utils.to_categorical(Y_encoded)
```

Splitting the dataset in 80-20 ratio

```
x_train, x_test, y_train, y_test = train_test_split(X, Y_final, test_size=0.2, r

print("Training Input shape\t: {}".format(x_train.shape))
print("Testing Input shape\t: {}".format(x_test.shape))
print("Training Output shape\t: {}".format(y_train.shape))
print("Testing Output shape\t: {}".format(y_test.shape))
```

```
Training Input shape    : (120, 4)
Testing Input shape     : (30, 4)
Training Output shape   : (120, 3)
Testing Output shape    : (30, 3)
```

Standardizing the dataset

```
std_clf = StandardScaler()
x_train_new = std_clf.fit_transform(x_train)
x_test_new = std_clf.transform(x_test)
```

1. If your target labels are ***one-hot encoded*** (binary vectors), use ***CategoricalCrossentropy***.
2. If your target labels are ***integer-encoded*** (class indices), use ***SparseCategoricalCrossentropy***

```
# Create and compile the model
model = Sequential()
model.add(Dense(12, input_dim=4, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Use this crossentropy loss function when there are two or more label classes
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur

# Train the model

model.fit(x_train_new, y_train, epochs=50, batch_size=7)
# Evaluate the model on the test data
loss, accuracy = model.evaluate(x_test_new, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/50
18/18 [==============================] - 1s 2ms/step - loss: 1.0237 - accur
Epoch 2/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9829 - accur
Epoch 3/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9452 - accur
Epoch 4/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9051 - accur
Epoch 5/50
18/18 [==============================] - 0s 2ms/step - loss: 0.8617 - accur
Epoch 6/50
18/18 [==============================] - 0s 2ms/step - loss: 0.8170 - accur
```

```
Epoch 7/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7679 - accur
Epoch 8/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7155 - accur
Epoch 9/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6693 - accur
Epoch 10/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6205 - accur
Epoch 11/50
18/18 [==============================] - 0s 2ms/step - loss: 0.5734 - accur
Epoch 12/50
18/18 [==============================] - 0s 2ms/step - loss: 0.5293 - accur
Epoch 13/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4882 - accur
Epoch 14/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4456 - accur
Epoch 15/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4093 - accur
Epoch 16/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3821 - accur
Epoch 17/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3575 - accur
Epoch 18/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3334 - accur
Epoch 19/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3136 - accur
Epoch 20/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2944 - accur
Epoch 21/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2787 - accur
Epoch 22/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2659 - accur
Epoch 23/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2537 - accur
Epoch 24/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2402 - accur
Epoch 25/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2295 - accur
Epoch 26/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2211 - accur
Epoch 27/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2097 - accur
Epoch 28/50
18/18 [==============================] - 0s 4ms/step - loss: 0.2031 - accur
Epoch 29/50
18/18 [==============================] - 0s 4ms/step - loss: 0.1978 - accur
```

3- Example 3 : IRIS dataset with one hot encoder

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder,OneHotEncoder
from sklearn.datasets import load_iris
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
from keras.utils import to_categorical


 #Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)


dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal-length  150 non-null    float64
 1   sepal-width   150 non-null    float64
 2   petal-length  150 non-null    float64
 3   petal-width   150 non-null    float64
 4   class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
Y = dataset['class']
X = dataset.drop(['class'], axis=1)
print("Shape of Input  features: {}".format(X.shape))
print("Shape of Output features: {}".format(Y.shape))
```

```
Shape of Input  features: (150, 4)
Shape of Output features: (150,)
```

```
X.head()
```

```
Y.head()
```

```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: class, dtype: object
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
import pandas as pd

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_

# Convert Pandas Series to NumPy arrays
y_train_array = y_train.to_numpy().reshape(-1, 1)
y_test_array = y_test.to_numpy().reshape(-1, 1)

# Scale features using MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# One-hot encode target values
one_hot = OneHotEncoder(sparse=False)  # Use sparse=False to get a dense array

y_train_hot = one_hot.fit_transform(y_train_array)
y_test_hot = one_hot.transform(y_test_array)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:
  warnings.warn(
```

```
# Create and compile the model
model = Sequential()
model.add(Dense(12, input_dim=4, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur
model.fit(X_train_scaled, y_train_hot, epochs=50, batch_size=7)
# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test_scaled, y_test_hot
                                )
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/50
18/18 [==============================] - 1s 3ms/step - loss: 1.1227 - accur
Epoch 2/50
18/18 [==============================] - 0s 2ms/step - loss: 1.1008 - accur
Epoch 3/50
18/18 [==============================] - 0s 2ms/step - loss: 1.0816 - accur
Epoch 4/50
18/18 [==============================] - 0s 2ms/step - loss: 1.0618 - accur
Epoch 5/50
18/18 [==============================] - 0s 2ms/step - loss: 1.0420 - accur
Epoch 6/50
18/18 [==============================] - 0s 2ms/step - loss: 1.0211 - accur
Epoch 7/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9996 - accur
Epoch 8/50
```

```
Epoch 8/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9752 - accur
Epoch 9/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9497 - accur
Epoch 10/50
18/18 [==============================] - 0s 2ms/step - loss: 0.9239 - accur
Epoch 11/50
18/18 [==============================] - 0s 3ms/step - loss: 0.8960 - accur
Epoch 12/50
18/18 [==============================] - 0s 2ms/step - loss: 0.8681 - accur
Epoch 13/50
18/18 [==============================] - 0s 2ms/step - loss: 0.8415 - accur
Epoch 14/50
18/18 [==============================] - 0s 2ms/step - loss: 0.8177 - accur
Epoch 15/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7952 - accur
Epoch 16/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7777 - accur
Epoch 17/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7565 - accur
Epoch 18/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7395 - accur
Epoch 19/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7232 - accur
Epoch 20/50
18/18 [==============================] - 0s 2ms/step - loss: 0.7096 - accur
Epoch 21/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6976 - accur
Epoch 22/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6831 - accur
Epoch 23/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6707 - accur
Epoch 24/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6602 - accur
Epoch 25/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6481 - accur
Epoch 26/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6357 - accur
Epoch 27/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6248 - accur
Epoch 28/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6111 - accur
Epoch 29/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6010 - accur
```

4- Example 4 : IRIS dataset with your own Model