Keras/Tensorflow Modelling Tutorial: Iris Dataset

This tutorial will demonstrate using Keras for deep learning on the Iris dataset. We will also go through the following:

- Creating data and target matrices from imported data
- One hot encoding of non-numerical class labels
- Building a convolutional neural network (CNN) model
- Accuracy after training and testing the model
- Depiction of confusion matrix

This code is adapted from "Multi-Class Classification Tutorial with the Keras Deep Learning Library" from Machine Learning Mastery and "Deep Learning Iris Dataset Keras" from Kaggle. The links are included below:

- https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
- https://www.kaggle.com/akashsri99/deep-learning-iris-dataset-keras

Importing Libraries

In [1]:

```
import warnings; warnings.simplefilter('ignore')

import os
os.environ['KERAS_BACKEND'] = 'tensorflow'

from keras.backend import set_image_dim_ordering
set_image_dim_ordering('tf')
```

```
Using TensorFlow backend.
```

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from seaborn import heatmap
%matplotlib inline
```

In [3]:

```
from keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD,Adam
```

In [4]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
```

Creating data (X) and target (Y) matrices

In [5]:

```
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", heade
r=None)
df.head()
```

Out[5]:

	0	1	2	3	4

	0	1	2	3	4
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

In [6]:

```
dataset = df.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

In [7]:

```
X.shape, Y.shape
```

Out[7]:

```
((150, 4), (150,))
```

One Hot Encoding

In "Keras/Tensorflow Modelling Tutorial: MNIST Dataset" tutorial from the class, we have used one hot encoding on numerical target data (0-9). In this case, the targets are words (names of the irises). We can use the builtin LabelEncoder function to first transform the words into numbers, then take the numbers to create matrix representations.

In [8]:

```
encoder = LabelEncoder()
```

In [9]:

```
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
encoded_Y
```

Out[9]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [16]:

```
y = np_utils.to_categorical(encoded_Y)
y[1::10]
```

Out[16]:

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
```

```
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]], dtype=float32)
```

Building Learning Model

For this example, we will build a one layer network with 4 inputs, 8 hidden nodes, and 3 outputs for demonstration.

In [17]:

```
from sklearn.model_selection import train_test_split
```

In [18]:

```
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=8)
```

In [38]:

```
model = Sequential()
```

In [39]:

```
model.add(Dense(8,input_shape=(4,),activation='relu'))
model.add(Dense(3,activation='softmax'))
```

In [40]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_16 (Dense)             (None, 8)                 40        
_________________________________________________________________
dense_17 (Dense)             (None, 3)                 27        
=================================================================
Total params: 67
Trainable params: 67
Non-trainable params: 0
_________________________________________________________________
```

In [41]:

```
model.compile(Adam(lr=0.04),'categorical_crossentropy',metrics=['accuracy'])
```

In [42]:

```
history = model.fit(X_train,y_train,epochs=100)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0801 03:27:00.197965 140735736558528 deprecation.py:323] From
/Users/xiaosg/anaconda3/lib/python3.6/site-packages/tensorflow/python/ops/math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Train on 105 samples
Epoch 1/100
105/105 [==============================] - 0s 2ms/sample - loss: 1.5491 - accuracy: 0.2667
Epoch 2/100
105/105 [==============================] - 0s 92us/sample - loss: 1.0354 - accuracy: 0.4571
Epoch 3/100
105/105 [==============================] - 0s 121us/sample - loss: 0.8507 - accuracy: 0.6571
Epoch 4/100
105/105 [==============================] - 0s 152us/sample - loss: 0.5320 - accuracy: 0.7429
```

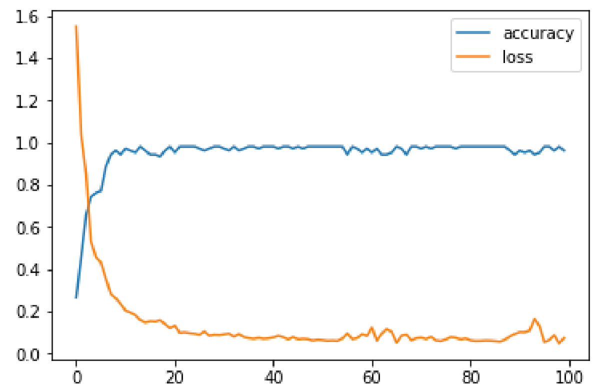
```
Epoch 5/100
105/105 [==============================] - 0s 162us/sample - loss: 0.4578 - accuracy: 0.7619
Epoch 6/100
105/105 [==============================] - 0s 131us/sample - loss: 0.4315 - accuracy: 0.7714
Epoch 7/100
105/105 [==============================] - 0s 98us/sample - loss: 0.3555 - accuracy: 0.8857
Epoch 8/100
105/105 [==============================] - 0s 134us/sample - loss: 0.2837 - accuracy: 0.9429
Epoch 9/100
105/105 [==============================] - 0s 127us/sample - loss: 0.2634 - accuracy: 0.9619
Epoch 10/100
105/105 [==============================] - 0s 157us/sample - loss: 0.2367 - accuracy: 0.9429
Epoch 11/100
105/105 [==============================] - 0s 136us/sample - loss: 0.2044 - accuracy: 0.9714
Epoch 12/100
105/105 [==============================] - 0s 145us/sample - loss: 0.1960 - accuracy: 0.9619
Epoch 13/100
105/105 [==============================] - 0s 166us/sample - loss: 0.1825 - accuracy: 0.9524
Epoch 14/100
105/105 [==============================] - 0s 158us/sample - loss: 0.1604 - accuracy: 0.9810
Epoch 15/100
105/105 [==============================] - 0s 151us/sample - loss: 0.1487 - accuracy: 0.9619
Epoch 16/100
105/105 [==============================] - 0s 162us/sample - loss: 0.1550 - accuracy: 0.9429
Epoch 17/100
105/105 [==============================] - 0s 158us/sample - loss: 0.1531 - accuracy: 0.9429
Epoch 18/100
105/105 [==============================] - 0s 130us/sample - loss: 0.1598 - accuracy: 0.9333
Epoch 19/100
105/105 [==============================] - 0s 183us/sample - loss: 0.1417 - accuracy: 0.9619
Epoch 20/100
105/105 [==============================] - 0s 146us/sample - loss: 0.1225 - accuracy: 0.9810
Epoch 21/100
105/105 [==============================] - 0s 194us/sample - loss: 0.1347 - accuracy: 0.9524
Epoch 22/100
105/105 [==============================] - 0s 160us/sample - loss: 0.0998 - accuracy: 0.9810
Epoch 23/100
105/105 [==============================] - ETA: 0s - loss: 0.1129 - accuracy: 0.96 - 0s 
160us/sample - loss: 0.1023 - accuracy: 0.9810
Epoch 24/100
105/105 [==============================] - 0s 150us/sample - loss: 0.0983 - accuracy: 0.9810
Epoch 25/100
105/105 [==============================] - 0s 217us/sample - loss: 0.0960 - accuracy: 0.9810
Epoch 26/100
105/105 [==============================] - 0s 173us/sample - loss: 0.0904 - accuracy: 0.9714
Epoch 27/100
105/105 [==============================] - 0s 140us/sample - loss: 0.1065 - accuracy: 0.9619
Epoch 28/100
105/105 [==============================] - 0s 117us/sample - loss: 0.0864 - accuracy: 0.9714
Epoch 29/100
105/105 [==============================] - 0s 175us/sample - loss: 0.0913 - accuracy: 0.9810
Epoch 30/100
105/105 [==============================] - 0s 159us/sample - loss: 0.0894 - accuracy: 0.9810
Epoch 31/100
105/105 [==============================] - 0s 224us/sample - loss: 0.0932 - accuracy: 0.9714
Epoch 32/100
105/105 [==============================] - 0s 167us/sample - loss: 0.0965 - accuracy: 0.9619
Epoch 33/100
105/105 [==============================] - 0s 177us/sample - loss: 0.0818 - accuracy: 0.9810
Epoch 34/100
105/105 [==============================] - 0s 202us/sample - loss: 0.0948 - accuracy: 0.9619
Epoch 35/100
105/105 [==============================] - 0s 192us/sample - loss: 0.0801 - accuracy: 0.9714
Epoch 36/100
105/105 [==============================] - 0s 200us/sample - loss: 0.0755 - accuracy: 0.9810
Epoch 37/100
105/105 [==============================] - 0s 173us/sample - loss: 0.0718 - accuracy: 0.9810
Epoch 38/100
105/105 [==============================] - 0s 187us/sample - loss: 0.0768 - accuracy: 0.9714
Epoch 39/100
105/105 [==============================] - 0s 181us/sample - loss: 0.0722 - accuracy: 0.9810
Epoch 40/100
105/105 [==============================] - 0s 284us/sample - loss: 0.0745 - accuracy: 0.9810
Epoch 41/100
105/105 [==============================] - 0s 172us/sample - loss: 0.0784 - accuracy: 0.9810
Epoch 42/100
105/105 [==============================] - 0s 148us/sample - loss: 0.0863 - accuracy: 0.9714
```

```
Epoch 43/100
105/105 [==============================] - 0s 151us/sample - loss: 0.0794 - accuracy: 0.9810
Epoch 44/100
105/105 [==============================] - 0s 184us/sample - loss: 0.0696 - accuracy: 0.9810
Epoch 45/100
105/105 [==============================] - ETA: 0s - loss: 0.1051 - accuracy: 0.96 - 0s
164us/sample - loss: 0.0805 - accuracy: 0.9714
Epoch 46/100
105/105 [==============================] - 0s 172us/sample - loss: 0.0695 - accuracy: 0.9810
Epoch 47/100
105/105 [==============================] - 0s 190us/sample - loss: 0.0719 - accuracy: 0.9714
Epoch 48/100
105/105 [==============================] - 0s 147us/sample - loss: 0.0701 - accuracy: 0.9810
Epoch 49/100
105/105 [==============================] - 0s 139us/sample - loss: 0.0635 - accuracy: 0.9810
Epoch 50/100
105/105 [==============================] - ETA: 0s - loss: 0.1310 - accuracy: 0.96 - 0s
178us/sample - loss: 0.0674 - accuracy: 0.9810
Epoch 51/100
105/105 [==============================] - 0s 206us/sample - loss: 0.0652 - accuracy: 0.9810
Epoch 52/100
105/105 [==============================] - 0s 208us/sample - loss: 0.0623 - accuracy: 0.9810
Epoch 53/100
105/105 [==============================] - 0s 229us/sample - loss: 0.0633 - accuracy: 0.9810
Epoch 54/100
105/105 [==============================] - 0s 179us/sample - loss: 0.0618 - accuracy: 0.9810
Epoch 55/100
105/105 [==============================] - 0s 166us/sample - loss: 0.0734 - accuracy: 0.9810
Epoch 56/100
105/105 [==============================] - 0s 217us/sample - loss: 0.0979 - accuracy: 0.9429
Epoch 57/100
105/105 [==============================] - 0s 169us/sample - loss: 0.0703 - accuracy: 0.9810
Epoch 58/100
105/105 [==============================] - 0s 177us/sample - loss: 0.0753 - accuracy: 0.9714
Epoch 59/100
105/105 [==============================] - 0s 131us/sample - loss: 0.0942 - accuracy: 0.9524
Epoch 60/100
105/105 [==============================] - 0s 246us/sample - loss: 0.0859 - accuracy: 0.9714
Epoch 61/100
105/105 [==============================] - 0s 224us/sample - loss: 0.1255 - accuracy: 0.9524
Epoch 62/100
105/105 [==============================] - 0s 198us/sample - loss: 0.0627 - accuracy: 0.9714
Epoch 63/100
105/105 [==============================] - 0s 228us/sample - loss: 0.0963 - accuracy: 0.9429
Epoch 64/100
105/105 [==============================] - 0s 251us/sample - loss: 0.1181 - accuracy: 0.9429
Epoch 65/100
105/105 [==============================] - 0s 220us/sample - loss: 0.1049 - accuracy: 0.9524
Epoch 66/100
105/105 [==============================] - 0s 296us/sample - loss: 0.0535 - accuracy: 0.9810
Epoch 67/100
105/105 [==============================] - 0s 189us/sample - loss: 0.0868 - accuracy: 0.9714
Epoch 68/100
105/105 [==============================] - 0s 256us/sample - loss: 0.0919 - accuracy: 0.9429
Epoch 69/100
105/105 [==============================] - 0s 217us/sample - loss: 0.0648 - accuracy: 0.9810
Epoch 70/100
105/105 [==============================] - 0s 184us/sample - loss: 0.0751 - accuracy: 0.9810
Epoch 71/100
105/105 [==============================] - 0s 229us/sample - loss: 0.0775 - accuracy: 0.9714
Epoch 72/100
105/105 [==============================] - 0s 301us/sample - loss: 0.0706 - accuracy: 0.9810
Epoch 73/100
105/105 [==============================] - 0s 112us/sample - loss: 0.0813 - accuracy: 0.9714
Epoch 74/100
105/105 [==============================] - 0s 96us/sample - loss: 0.0643 - accuracy: 0.9810
Epoch 75/100
105/105 [==============================] - 0s 91us/sample - loss: 0.0610 - accuracy: 0.9810
Epoch 76/100
105/105 [==============================] - 0s 166us/sample - loss: 0.0690 - accuracy: 0.9810
Epoch 77/100
105/105 [==============================] - 0s 147us/sample - loss: 0.0800 - accuracy: 0.9810
Epoch 78/100
105/105 [==============================] - ETA: 0s - loss: 0.0707 - accuracy: 0.96 - 0s
152us/sample - loss: 0.0770 - accuracy: 0.9714
Epoch 79/100
105/105 [==============================] - 0s 129us/sample - loss: 0.0689 - accuracy: 0.9810
```

```
Epoch 80/100
105/105 [==============================] - 0s 128us/sample - loss: 0.0736 - accuracy: 0.9810
Epoch 81/100
105/105 [==============================] - 0s 110us/sample - loss: 0.0643 - accuracy: 0.9810
Epoch 82/100
105/105 [==============================] - 0s 145us/sample - loss: 0.0616 - accuracy: 0.9810
Epoch 83/100
105/105 [==============================] - 0s 123us/sample - loss: 0.0617 - accuracy: 0.9810
Epoch 84/100
105/105 [==============================] - 0s 235us/sample - loss: 0.0634 - accuracy: 0.9810
Epoch 85/100
105/105 [==============================] - 0s 142us/sample - loss: 0.0629 - accuracy: 0.9810
Epoch 86/100
105/105 [==============================] - 0s 150us/sample - loss: 0.0607 - accuracy: 0.9810
Epoch 87/100
105/105 [==============================] - 0s 152us/sample - loss: 0.0586 - accuracy: 0.9810
Epoch 88/100
105/105 [==============================] - 0s 117us/sample - loss: 0.0674 - accuracy: 0.9810
Epoch 89/100
105/105 [==============================] - 0s 128us/sample - loss: 0.0830 - accuracy: 0.9619
Epoch 90/100
105/105 [==============================] - 0s 154us/sample - loss: 0.0950 - accuracy: 0.9429
Epoch 91/100
105/105 [==============================] - 0s 92us/sample - loss: 0.1034 - accuracy: 0.9619
Epoch 92/100
105/105 [==============================] - 0s 154us/sample - loss: 0.1028 - accuracy: 0.9524
Epoch 93/100
105/105 [==============================] - 0s 134us/sample - loss: 0.1106 - accuracy: 0.9619
Epoch 94/100
105/105 [==============================] - 0s 144us/sample - loss: 0.1662 - accuracy: 0.9429
Epoch 95/100
105/105 [==============================] - 0s 152us/sample - loss: 0.1325 - accuracy: 0.9524
Epoch 96/100
105/105 [==============================] - 0s 141us/sample - loss: 0.0561 - accuracy: 0.9810
Epoch 97/100
105/105 [==============================] - 0s 90us/sample - loss: 0.0664 - accuracy: 0.9810
Epoch 98/100
105/105 [==============================] - 0s 111us/sample - loss: 0.0879 - accuracy: 0.9619
Epoch 99/100
105/105 [==============================] - 0s 95us/sample - loss: 0.0499 - accuracy: 0.9810
Epoch 100/100
105/105 [==============================] - 0s 138us/sample - loss: 0.0760 - accuracy: 0.9619
```

In [43]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['loss'], label='loss')
plt.legend()
plt.show()
```



Evaluation of Learning Model

In [44]:

```
y_hat = model.predict(X_test)
```

In [45]:

```
model.evaluate(X_test, y_test)
```

```
45/45 [==============================] - 0s 911us/sample - loss: 0.0628 - accuracy: 0.9556
```

Out[45]:

```
[0.06283027674588892, 0.95555556]
```

Confusion Matrix

In [46]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

Confusion matrix C is such that c(i,j) is equal to the number of observations known to be in group i but predicted to be in group j.

In [47]:

```
y_test_class = np.argmax(y_test,axis=1)
y_pred_class = np.argmax(y_hat,axis=1)
```

In [48]:

```
print(classification_report(y_test_class,y_pred_class))
C = confusion_matrix(y_test_class,y_pred_class)
print(C)
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        15
          1       0.94      0.94      0.94        16
          2       0.93      0.93      0.93        14

avg / total       0.96      0.96      0.96        45

[[15  0  0]
 [ 0 15  1]
 [ 0  1 13]]
```

In [49]:

```
heatmap(C, square=True, annot=True, cbar=False, cmap="YlGnBu")
plt.xlabel('predicted value')
plt.ylabel('true value')
```

Out[49]:

```
Text(91.68,0.5,'true value')
```