using deep learnig on iris data set to find best accurasy

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
#to load dataset
df=pd.read_csv("/content/drive/MyDrive/DEEP LEARNING/iris.csv")
```

```
#To show first 5 records
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
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

```
#To check null values
df.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

```
#To check datatypes
df.dtypes
```

```
sepal_length    float64
```

```
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

```
#To check duplicates row
df.duplicated().sum()
```

```
3
```

```
#To delete duplicates row permanently
df.drop_duplicates(inplace=True)
```

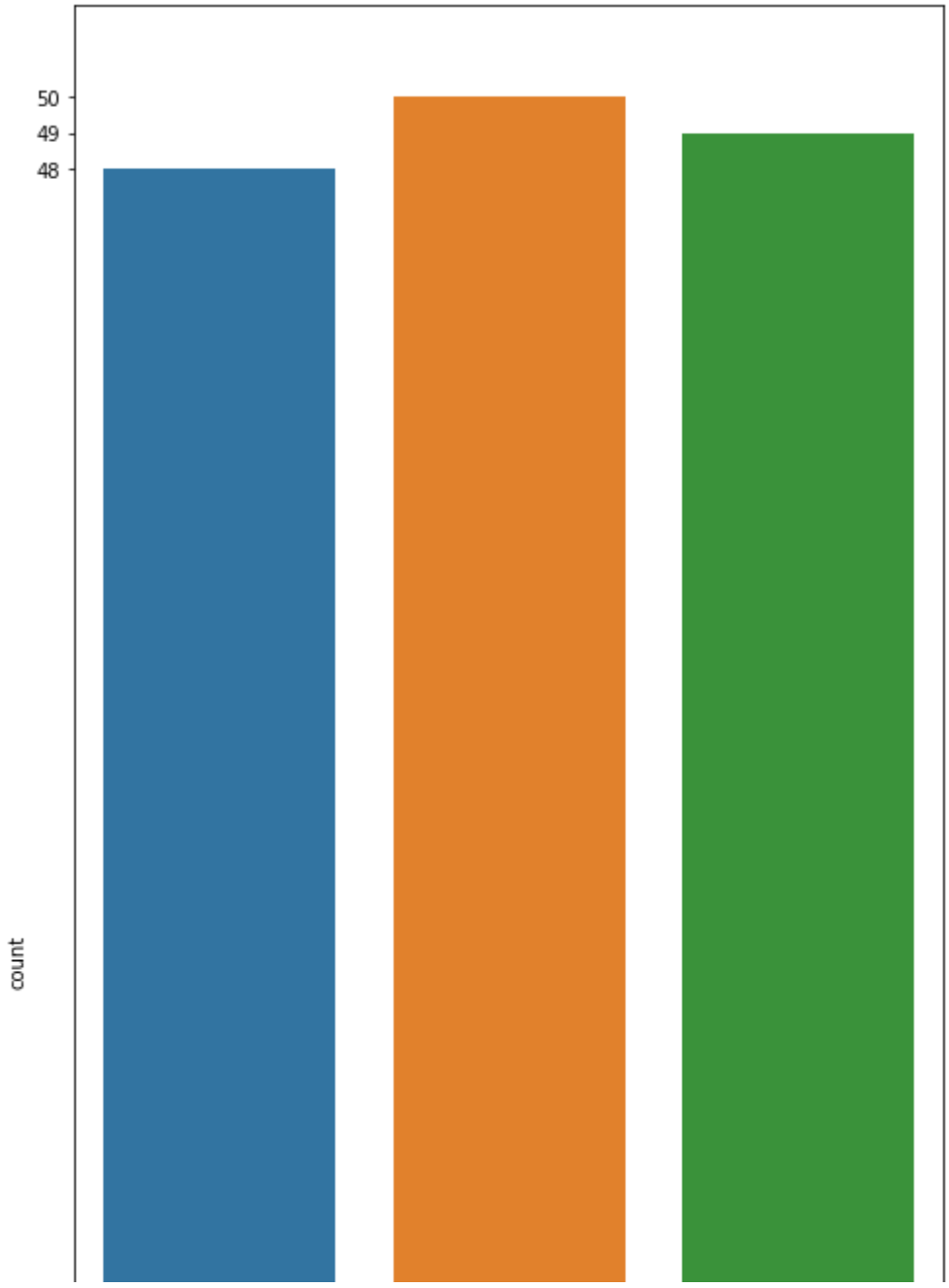
```
#To check duplicates row
df.duplicated().sum()
```

```
0
```

```
#How many labels/classes
df['species'].value_counts()
```

```
Iris-versicolor    50
Iris-virginica     49
Iris-setosa        48
Name: species, dtype: int64
```

```
plt.figure(figsize=(8,18)) #width,height
sns.countplot(data=df,x="species")
f=df['species'].value_counts()
plt.yticks(f)
plt.show()
```




```
#convert object type column species into number with the help of LabelEncoder
from sklearn.preprocessing import LabelEncoder
#create object of LabelEncoder class
le=LabelEncoder()
df['species']=le.fit_transform(df['species'])
df.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species           int64
dtype: object
```

```
#select input and output from dataset
x=df.drop('species',axis=1)
y=df['species']
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1,stratify=y
```

```
y_train.value_counts()
```

```
1    35
2    34
0    33
Name: species, dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)
```

```
import tensorflow as tf
```

```
#create modelmof sequential class
model=tf.keras.models.Sequential([
       tf.keras.layers.Dense(units=4,activation="relu",input_shape=(x.shape[1],)),#hidden
       tf.keras.layers.Dense(units=4,activation="relu"),#hidden layer 2
       tf.keras.layers.Dense(units=3,activation="softmax")])#output
```

```
model.summary()#param=no.of new rowa*no of input+bias
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 20

 dense_1 (Dense)             (None, 4)                 20

 dense_2 (Dense)             (None, 3)                 15

=================================================================
Total params: 55
Trainable params: 55
Non-trainable params: 0
_________________________________________________________________
```

```
#compile the model
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
#Early Stopping :
from tensorflow.keras.callbacks import EarlyStopping
#callbacks inbuilt parameter of fit()
#create callback : -
#EarlyStopping() inbuilt function
callback=EarlyStopping(
    monitor="val_loss",  #val_loss means testing error
```

```
    min_delta=0.00001, #value of lambda
    patience=20,
    verbose=1,
    mode="auto", #min loss
    baseline=None,
    restore_best_weights=False
)

#train the model use inbuilt method fit()
trained_model=model.fit(x_train,y_train,epochs=3500,validation_data=(x_test,y_test),callba
```

```
Epoch 1/3500
4/4 [==============================] - 2s 182ms/step - loss: 1.4566 - accuracy: 0.
Epoch 2/3500
4/4 [==============================] - 0s 33ms/step - loss: 1.4011 - accuracy: 0.2
Epoch 3/3500
4/4 [==============================] - 0s 22ms/step - loss: 1.3430 - accuracy: 0.3
Epoch 4/3500
4/4 [==============================] - 0s 53ms/step - loss: 1.2855 - accuracy: 0.3
Epoch 5/3500
4/4 [==============================] - 0s 41ms/step - loss: 1.2514 - accuracy: 0.2
Epoch 6/3500
4/4 [==============================] - 0s 38ms/step - loss: 1.2246 - accuracy: 0.3
Epoch 7/3500
4/4 [==============================] - 0s 45ms/step - loss: 1.1958 - accuracy: 0.3
Epoch 8/3500
4/4 [==============================] - 0s 58ms/step - loss: 1.1679 - accuracy: 0.3
Epoch 9/3500
4/4 [==============================] - 0s 17ms/step - loss: 1.1488 - accuracy: 0.3
Epoch 10/3500
4/4 [==============================] - 0s 28ms/step - loss: 1.1261 - accuracy: 0.3
Epoch 11/3500
4/4 [==============================] - 0s 23ms/step - loss: 1.1076 - accuracy: 0.3
Epoch 12/3500
4/4 [==============================] - 0s 29ms/step - loss: 1.0895 - accuracy: 0.3
Epoch 13/3500
4/4 [==============================] - 0s 31ms/step - loss: 1.0735 - accuracy: 0.5
Epoch 14/3500
4/4 [==============================] - 0s 46ms/step - loss: 1.0584 - accuracy: 0.6
Epoch 15/3500
4/4 [==============================] - 0s 32ms/step - loss: 1.0445 - accuracy: 0.7
Epoch 16/3500
4/4 [==============================] - 0s 25ms/step - loss: 1.0286 - accuracy: 0.7
Epoch 17/3500
4/4 [==============================] - 0s 35ms/step - loss: 1.0133 - accuracy: 0.7
Epoch 18/3500
4/4 [==============================] - 0s 31ms/step - loss: 1.0010 - accuracy: 0.7
Epoch 19/3500
4/4 [==============================] - 0s 33ms/step - loss: 0.9877 - accuracy: 0.7
Epoch 20/3500
4/4 [==============================] - 0s 29ms/step - loss: 0.9749 - accuracy: 0.7
Epoch 21/3500
4/4 [==============================] - 0s 25ms/step - loss: 0.9625 - accuracy: 0.7
Epoch 22/3500
4/4 [==============================] - 0s 38ms/step - loss: 0.9494 - accuracy: 0.8
Epoch 23/3500
4/4 [==============================] - 0s 27ms/step - loss: 0.9387 - accuracy: 0.7
Epoch 24/3500
4/4 [==============================] - 0s 35ms/step - loss: 0.9265 - accuracy: 0.7
```

```
Epoch 25/3500
4/4 [==============================] - 0s 55ms/step - loss: 0.9166 - accuracy: 0.8
Epoch 26/3500
4/4 [==============================] - 0s 30ms/step - loss: 0.9069 - accuracy: 0.8
Epoch 27/3500
4/4 [==============================] - 0s 38ms/step - loss: 0.8965 - accuracy: 0.8
Epoch 28/3500
4/4 [==============================] - 0s 25ms/step - loss: 0.8849 - accuracy: 0.8
Epoch 29/3500
```

```
#traning
model.evaluate(x_train,y_train)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.0795 - accuracy: 0.9706
[0.07948414236307144, 0.970588207244873]
```

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

```
2/2 [==============================] - 0s 9ms/step - loss: 0.0900 - accuracy: 0.9778
[0.08998732268810272, 0.9777777791023254]
```

```
#find the prediction means test the model
y_pred=model.predict(x_test).round(2)
y_pred
```

```
array([[0.02, 0.05, 0.93],
       [0.01, 0.79, 0.21],
       [0.02, 0.07, 0.91],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  ],
       [0.01, 0.94, 0.05],
       [0.  , 0.94, 0.06],
       [1.  , 0.  , 0.  ],
       [0.02, 0.05, 0.93],
       [0.  , 0.99, 0.01],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [0.  , 0.99, 0.01],
       [0.02, 0.29, 0.69],
       [0.01, 0.99, 0.01],
       [0.  , 1.  , 0.  ],
       [0.03, 0.62, 0.35],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  ],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  ],
       [0.02, 0.05, 0.93],
       [1.  , 0.  , 0.  ],
       [0.  , 0.98, 0.01],
```

```
       [0.  , 0.99, 0.01],
       [1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  ],
       [0.  , 0.98, 0.01],
       [0.02, 0.05, 0.93],
       [0.02, 0.29, 0.69],
       [0.01, 0.92, 0.07],
       [1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  ],
       [0.02, 0.05, 0.93],
       [0.  , 0.99, 0.01],
       [0.  , 0.99, 0.01],
       [1.  , 0.  , 0.  ],
       [0.02, 0.05, 0.93]], dtype=float32)
```

```
#list comprehension
y_pred=[np.argmax(i) for i in y_pred]
y_pred
```

```
[2,
 1,
 2,
 2,
 0,
 0,
 1,
 1,
 1,
 0,
 2,
 1,
 2,
 0,
 1,
 2,
 1,
 1,
 1,
 2,
 0,
 2,
 0,
 0,
 1,
 2,
 0,
 0,
 2,
 0,
 1,
 1,
 0,
 0,
 1,
 2,
 2,
 1,
 0,
 0,
```

```
       2,
       1,
       1,
       0,
       2]
```

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

```
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00        15
               1       0.94      1.00      0.97        15
               2       1.00      0.93      0.97        15

        accuracy                           0.98        45
       macro avg       0.98      0.98      0.98        45
    weighted avg       0.98      0.98      0.98        45

    [[15  0  0]
     [ 0 15  0]
     [ 0  1 14]]
```