**Name: Abishek Kumar**

**Reg. No: CU23MCA001A**

**Full file with ouput is available on github:**

[*https://github.com/Abishek-Kumar-GHub/Tensorflow-Functions*](https://github.com/Abishek-Kumar-GHub/Tensorflow-Functions)

***Mainfile:***[*tensor-flow2.ipynb*](https://github.com/Abishek-Kumar-GHub/Tensorflow-Functions/blob/main/tensor-flow2.ipynb)

**Topic:** Tensor Flow

**Implemented:** 20 functions

**Dataset:** Abalone, Iris (.csv File)

**Dependencies Needed:** tensorflow, sci-kit learn, pandas, numpy, os, matplotlib

**Code:**

*# Import necessary libraries*

import tensorflow as tf

from tensorflow.keras.layers import Dense, Input, Dropout, BatchNormalization

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard

from tensorflow.keras.regularizers import l1, l2

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

import pandas as pd

import numpy as np

import os  
  
***Load Abalone Dataset (Regression)***

abalone\_df = pd.read\_csv('abalone.csv')

abalone\_df['Sex'] = abalone\_df['Sex'].map({'M': 0, 'F': 1, 'I': 2}) # Encode categorical column

X\_abalone = abalone\_df.drop(columns=['Rings']).values

y\_abalone = abalone\_df['Rings'].values

***Split data into train/test sets***

X\_train\_abalone, X\_test\_abalone, y\_train\_abalone, y\_test\_abalone = train\_test\_split(X\_abalone, y\_abalone, test\_size=0.2, random\_state=42)

***Load Iris Dataset (Classification)***

iris = load\_iris()

X\_iris = iris.data

y\_iris = iris.target

*# Split Iris data into train/test sets*

X\_train\_iris, X\_test\_iris, y\_train\_iris, y\_test\_iris = train\_test\_split(X\_iris, y\_iris, test\_size=0.2, random\_state=42)

***1. Create TensorFlow Data Pipeline***

def preprocess(features, labels):

features = tf.cast(features, dtype=tf.float32)

labels = tf.cast(labels, dtype=tf.float32)

return features, labels

train\_dataset\_abalone = tf.data.Dataset.from\_tensor\_slices((X\_train\_abalone, y\_train\_abalone))

train\_dataset\_abalone = train\_dataset\_abalone.shuffle(buffer\_size=1024).batch(32).map(preprocess)

test\_dataset\_abalone = tf.data.Dataset.from\_tensor\_slices((X\_test\_abalone, y\_test\_abalone))

test\_dataset\_abalone = test\_dataset\_abalone.batch(32).map(preprocess)

***2. Simple Linear Regression Model for Abalone***

linear\_model\_abalone = Sequential([

Dense(1, input\_dim=X\_train\_abalone.shape[1], activation='linear') *# No activation for linear output*

])

linear\_model\_abalone.compile(optimizer='adam', loss='mse')

linear\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone))

***Output:***

Epoch 1/50

/home/abishek/.local/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 97.4211 - val\_loss: 92.3768

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 89.4056 - val\_loss: 84.2177

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 82.4970 - val\_loss: 76.6534

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 75.4907 - val\_loss: 69.7425

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 67.1278 - val\_loss: 63.4394

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 60.9302 - val\_loss: 57.6549

Epoch 7/50

.....

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 9.4066 - val\_loss: 9.6089

Epoch 45/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.3538 - val\_loss: 9.5084

Epoch 46/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 8.7337 - val\_loss: 9.4126

Epoch 47/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 9.0581 - val\_loss: 9.3237

Epoch 48/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 9.2692 - val\_loss: 9.2395

Epoch 49/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 8.9712 - val\_loss: 9.1640

Epoch 50/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 9.1860 - val\_loss: 9.0903

***3. Multi-layer Perceptron (MLP) Model for Abalone***

mlp\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu'),

Dense(64, activation='relu'),

Dense(1) # Linear output for regression

])

***Output:***

mlp\_model\_abalone.compile(optimizer='adam', loss='mse')

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

mlp\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 69.2738 - val\_loss: 8.4989

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.8506 - val\_loss: 7.1418

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.2806 - val\_loss: 6.4502

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.2701 - val\_loss: 6.0124

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.9721 - val\_loss: 5.7205

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.1073 - val\_loss: 5.5145

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.4194 - val\_loss: 5.3534

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.1688 - val\_loss: 5.2284

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.0647 - val\_loss: 5.0732

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 4.7196 - val\_loss: 4.9890

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.9673 - val\_loss: 4.9218

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.9999 - val\_loss: 4.8981

Epoch 13/50

...

Epoch 34/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.4207 - val\_loss: 4.5924

Epoch 35/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.7459 - val\_loss: 4.8502

***4. Dropout Model for Abalone***

dropout\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu'),

Dropout(0.5),

Dense(64, activation='relu'),

Dense(1)

])

dropout\_model\_abalone.compile(optimizer='adam', loss='mse')

dropout\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 63.5535 - val\_loss: 8.9383

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 10.1551 - val\_loss: 7.4329

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 8.6981 - val\_loss: 6.7528

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 7.8724 - val\_loss: 6.4108

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.6544 - val\_loss: 6.1528

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 7.3863 - val\_loss: 6.0635

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 7.0701 - val\_loss: 5.9063

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.1952 - val\_loss: 5.6544

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6.5318 - val\_loss: 5.5687

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6.6518 - val\_loss: 5.5087

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.3368 - val\_loss: 5.3268

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.2362 - val\_loss: 5.2445

Epoch 13/50

...

Epoch 39/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.2950 - val\_loss: 4.6671

Epoch 40/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.6877 - val\_loss: 4.6281

***5. Batch Normalization Model for Abalone***

batch\_norm\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu'),

BatchNormalization(),

Dense(64, activation='relu'),

Dense(1)

])

batch\_norm\_model\_abalone.compile(optimizer='adam', loss='mse')

batch\_norm\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 60.9734 - val\_loss: 52.8369

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 7.1260 - val\_loss: 40.0354

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.3929 - val\_loss: 23.9222

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.7844 - val\_loss: 12.1932

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.7713 - val\_loss: 7.3203

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.3094 - val\_loss: 4.8407

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.2988 - val\_loss: 4.5443

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 4.6529 - val\_loss: 4.5414

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 4.6911 - val\_loss: 4.4970

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 4.6882 - val\_loss: 7.7577

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.0638 - val\_loss: 4.5393

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.4863 - val\_loss: 5.1816

Epoch 13/50

...

Epoch 17/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 4.7558 - val\_loss: 4.5235

Epoch 18/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 4.3382 - val\_loss: 4.9553

***6. Custom Loss Function for Abalone***

def custom\_loss(y\_true, y\_pred):

return tf.reduce\_mean(tf.square(y\_true - y\_pred)) + 0.01 \* tf.reduce\_sum(tf.square(y\_pred))

***7. L1 Regularization Model for Abalone***

l1\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu', kernel\_regularizer=l1(0.01)),

Dense(64, activation='relu'),

Dense(1)

])

l1\_model\_abalone.compile(optimizer='adam', loss='mse')

l1\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 65.6926 - val\_loss: 9.9859

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.1381 - val\_loss: 8.1549

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.8076 - val\_loss: 7.2735

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 7.5869 - val\_loss: 6.7171

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6.3481 - val\_loss: 6.9097

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 6.5564 - val\_loss: 6.2688

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.1763 - val\_loss: 6.1027

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 5.8088 - val\_loss: 5.9631

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.6766 - val\_loss: 5.8310

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.2821 - val\_loss: 5.7629

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.6241 - val\_loss: 5.9015

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.4138 - val\_loss: 5.6592

Epoch 13/50

...

Epoch 49/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.7078 - val\_loss: 4.8690

Epoch 50/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 4.7245 - val\_loss: 5.0691

***8. L2 Regularization Model for Abalone***

l2\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu', kernel\_regularizer=l2(0.01)),

Dense(64, activation='relu'),

Dense(1)

])

l2\_model\_abalone.compile(optimizer='adam', loss='mse')

l2\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 59.6418 - val\_loss: 9.0235

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 7.9873 - val\_loss: 7.1768

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.9187 - val\_loss: 6.4324

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.9714 - val\_loss: 6.0286

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 5.6025 - val\_loss: 5.8654

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.3842 - val\_loss: 5.6954

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.5280 - val\_loss: 5.5761

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.1758 - val\_loss: 5.3277

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.0764 - val\_loss: 5.5336

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 5.4153 - val\_loss: 5.1587

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.0881 - val\_loss: 5.0792

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.2540 - val\_loss: 5.1598

Epoch 13/50

...

Epoch 30/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.6559 - val\_loss: 4.8649

Epoch 31/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.4894 - val\_loss: 4.7815

***9. Learning Rate Scheduling for Abalone***

lr\_scheduler\_abalone = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=3)

***10. Model Checkpointing for Abalone***

checkpoint\_abalone = ModelCheckpoint('best\_model\_abalone.keras', monitor='val\_loss', save\_best\_only=True)

***11. Train Model with TensorFlow Callbacks for Abalone***

model\_with\_callbacks\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu'),

Dense(64, activation='relu'),

Dense(1)

])

model\_with\_callbacks\_abalone.compile(optimizer='adam', loss='mse')

model\_with\_callbacks\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone),

callbacks=[early\_stopping, lr\_scheduler\_abalone, checkpoint\_abalone])

***Output:***

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 62.7525 - val\_loss: 8.4260 - learning\_rate: 0.0010

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 7.8807 - val\_loss: 6.9538 - learning\_rate: 0.0010

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.7699 - val\_loss: 6.1995 - learning\_rate: 0.0010

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.8205 - val\_loss: 5.8576 - learning\_rate: 0.0010

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.7159 - val\_loss: 5.5644 - learning\_rate: 0.0010

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.6225 - val\_loss: 5.4474 - learning\_rate: 0.0010

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 5.4462 - val\_loss: 5.2762 - learning\_rate: 0.0010

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5.1786 - val\_loss: 5.1296 - learning\_rate: 0.0010

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.1388 - val\_loss: 5.0556 - learning\_rate: 0.0010

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.7197 - val\_loss: 4.9468 - learning\_rate: 0.0010

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 4.9620 - val\_loss: 4.9126 - learning\_rate: 0.0010

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.9504 - val\_loss: 4.9212 - learning\_rate: 0.0010

Epoch 13/50

...

Epoch 49/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 4.2623 - val\_loss: 4.4940 - learning\_rate: 8.0000e-06

Epoch 50/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.0323 - val\_loss: 4.4942 - learning\_rate: 8.0000e-06

<keras.src.callbacks.history.History at 0x73c2859769e0>

***12. Custom Metric for Abalone: Mean Absolute Error***

class CustomMAE(tf.keras.metrics.Metric):

def \_\_init\_\_(self, name='custom\_mae', \*\*kwargs):

super(CustomMAE, self).\_\_init\_\_(name=name, \*\*kwargs)

self.total = self.add\_weight(name='total', initializer='zeros')

self.count = self.add\_weight(name='count', initializer='zeros')

def update\_state(self, y\_true, y\_pred, sample\_weight=None):

# Ensure both y\_true and y\_pred are of the same dtype

y\_true = tf.cast(y\_true, tf.float32) # Convert y\_true to float32 if it's int64

abs\_diff = tf.abs(y\_true - y\_pred)

self.total.assign\_add(tf.reduce\_sum(abs\_diff))

self.count.assign\_add(tf.cast(tf.size(y\_true), tf.float32))

def result(self):

return self.total / self.count

***13. Train with Custom Metric for Abalone***

custom\_metric\_model\_abalone = Sequential([

Dense(128, input\_dim=X\_train\_abalone.shape[1], activation='relu'),

Dense(64, activation='relu'),

Dense(1)

])

custom\_metric\_model\_abalone.compile(optimizer='adam', loss='mse', metrics=[CustomMAE()])

custom\_metric\_model\_abalone.fit(X\_train\_abalone, y\_train\_abalone, epochs=50, validation\_data=(X\_test\_abalone, y\_test\_abalone), callbacks=[early\_stopping])

*Output:*

Epoch 1/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - custom\_mae: 236.1168 - loss: 69.6725 - val\_custom\_mae: 95.9149 - val\_loss: 9.4854

Epoch 2/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - custom\_mae: 97.3454 - loss: 8.6560 - val\_custom\_mae: 98.2136 - val\_loss: 7.7414

Epoch 3/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - custom\_mae: 95.3377 - loss: 7.4309 - val\_custom\_mae: 92.3334 - val\_loss: 6.6468

Epoch 4/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - custom\_mae: 92.8815 - loss: 6.1663 - val\_custom\_mae: 91.2297 - val\_loss: 6.1600

Epoch 5/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - custom\_mae: 90.6498 - loss: 5.7749 - val\_custom\_mae: 91.5250 - val\_loss: 5.8707

Epoch 6/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - custom\_mae: 92.0013 - loss: 6.1814 - val\_custom\_mae: 91.6312 - val\_loss: 5.8204

Epoch 7/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - custom\_mae: 93.1652 - loss: 5.6069 - val\_custom\_mae: 94.8181 - val\_loss: 5.4694

Epoch 8/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - custom\_mae: 92.9937 - loss: 5.1976 - val\_custom\_mae: 99.8570 - val\_loss: 5.6010

Epoch 9/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - custom\_mae: 94.3242 - loss: 5.3531 - val\_custom\_mae: 95.1325 - val\_loss: 5.1574

Epoch 10/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - custom\_mae: 92.3160 - loss: 4.9372 - val\_custom\_mae: 97.1283 - val\_loss: 5.0624

Epoch 11/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - custom\_mae: 95.2191 - loss: 4.9103 - val\_custom\_mae: 94.6921 - val\_loss: 5.0480

Epoch 12/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - custom\_mae: 94.7513 - loss: 4.7421 - val\_custom\_mae: 99.6983 - val\_loss: 4.9265

Epoch 13/50

...

Epoch 36/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - custom\_mae: 94.3841 - loss: 4.6367 - val\_custom\_mae: 97.3900 - val\_loss: 4.5794

Epoch 37/50

105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - custom\_mae: 98.3031 - loss: 4.5658 - val\_custom\_mae: 97.1728 - val\_loss: 4.7111

<keras.src.callbacks.history.History at 0x73c292e849d0>

***14. TensorBoard for Visualization (Abalone)***

tensorboard\_callback\_abalone = TensorBoard(log\_dir=os.path.join('logs', 'abalone'))

***15. Predict with the Model (Abalone)***

sample\_data\_abalone = tf.convert\_to\_tensor(X\_test\_abalone[:5], dtype=tf.float32)

predictions\_abalone = model\_with\_callbacks\_abalone.predict(sample\_data\_abalone)

print("Sample Predictions (Abalone):", predictions\_abalone)

***Output:***

1/1━━━━━━━━━━━━━━━━━━━━0s 99ms/step

Sample Predictions (Abalone): [[12.142045 ]

[10.171386 ]

[14.602448 ]

[11.3944025]

[11.4544935]]

***Iris Dataset Classification Model (using MLP)***

***16. Multi-layer Perceptron (MLP) for Iris***

mlp\_model\_iris = Sequential([

Dense(128, input\_dim=X\_train\_iris.shape[1], activation='relu'),

Dense(64, activation='relu'),

Dense(3, activation='softmax') # 3 classes for classification (Iris dataset)

])

mlp\_model\_iris.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

mlp\_model\_iris.fit(X\_train\_iris, y\_train\_iris, epochs=50, validation\_data=(X\_test\_iris, y\_test\_iris), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 148ms/step - accuracy: 0.3498 - loss: 1.2284 - val\_accuracy: 0.6000 - val\_loss: 1.0492

Epoch 2/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.4625 - loss: 1.0390 - val\_accuracy: 0.3000 - val\_loss: 0.9768

Epoch 3/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.4202 - loss: 0.9314 - val\_accuracy: 0.8000 - val\_loss: 0.8216

Epoch 4/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.8671 - loss: 0.8141 - val\_accuracy: 0.7000 - val\_loss: 0.7324

Epoch 5/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6819 - loss: 0.7277 - val\_accuracy: 0.7667 - val\_loss: 0.6590

Epoch 6/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.8123 - loss: 0.6572 - val\_accuracy: 0.9000 - val\_loss: 0.6030

Epoch 7/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9262 - loss: 0.6020 - val\_accuracy: 0.9667 - val\_loss: 0.5693

Epoch 8/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9156 - loss: 0.5428 - val\_accuracy: 0.9667 - val\_loss: 0.5115

Epoch 9/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9354 - loss: 0.4971 - val\_accuracy: 0.8667 - val\_loss: 0.4671

Epoch 10/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.8329 - loss: 0.4826 - val\_accuracy: 0.8667 - val\_loss: 0.4373

Epoch 11/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8927 - loss: 0.4669 - val\_accuracy: 0.9667 - val\_loss: 0.4186

Epoch 12/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9529 - loss: 0.3991 - val\_accuracy: 0.9667 - val\_loss: 0.3828

Epoch 13/50

...

Epoch 49/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.9742 - loss: 0.1151 - val\_accuracy: 1.0000 - val\_loss: 0.1211

Epoch 50/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9792 - loss: 0.1135 - val\_accuracy: 1.0000 - val\_loss: 0.1187  
  
***17. Dropout for Iris Classification***

dropout\_model\_iris = Sequential([

Dense(128, input\_dim=X\_train\_iris.shape[1], activation='relu'),

Dropout(0.5),

Dense(64, activation='relu'),

Dense(3, activation='softmax')

])

dropout\_model\_iris.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

dropout\_model\_iris.fit(X\_train\_iris, y\_train\_iris, epochs=50, validation\_data=(X\_test\_iris, y\_test\_iris), callbacks=[early\_stopping])

***Output:***

Epoch 1/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 394ms/step - accuracy: 0.2965 - loss: 1.4251 - val\_accuracy: 0.3667 - val\_loss: 1.0739

Epoch 2/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 189ms/step - accuracy: 0.4056 - loss: 1.1186 - val\_accuracy: 0.3667 - val\_loss: 1.0320

Epoch 3/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 142ms/step - accuracy: 0.4081 - loss: 1.1379 - val\_accuracy: 0.3667 - val\_loss: 0.9232

Epoch 4/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.5348 - loss: 1.0219 - val\_accuracy: 0.7000 - val\_loss: 0.8526

Epoch 5/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step - accuracy: 0.5894 - loss: 0.9842 - val\_accuracy: 0.7000 - val\_loss: 0.8072

Epoch 6/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.6765 - loss: 0.8643 - val\_accuracy: 0.7000 - val\_loss: 0.7546

Epoch 7/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - accuracy: 0.6117 - loss: 0.8786 - val\_accuracy: 0.7000 - val\_loss: 0.7034

Epoch 8/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 143ms/step - accuracy: 0.6844 - loss: 0.7973 - val\_accuracy: 0.7000 - val\_loss: 0.6486

Epoch 9/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.7392 - loss: 0.7104 - val\_accuracy: 0.7000 - val\_loss: 0.6002

Epoch 10/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.6608 - loss: 0.7911 - val\_accuracy: 0.7000 - val\_loss: 0.5615

Epoch 11/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.7454 - loss: 0.7095 - val\_accuracy: 0.7000 - val\_loss: 0.5240

Epoch 12/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.7223 - loss: 0.6797 - val\_accuracy: 0.7000 - val\_loss: 0.4993

Epoch 13/50

...

Epoch 49/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - accuracy: 0.9592 - loss: 0.1643 - val\_accuracy: 0.9667 - val\_loss: 0.1506

Epoch 50/50

4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 253ms/step - accuracy: 0.9198 - loss: 0.1815 - val\_accuracy: 0.9667 - val\_loss: 0.1381

***18. Model Checkpointing for Iris***

checkpoint\_iris = ModelCheckpoint('best\_model\_iris.keras', monitor='val\_loss', save\_best\_only=True)

***19. Predict with the Iris Model***

sample\_data\_iris = tf.convert\_to\_tensor(X\_test\_iris[:5], dtype=tf.float32)

predictions\_iris = mlp\_model\_iris.predict(sample\_data\_iris)

print("Sample Predictions (Iris):", predictions\_iris)

***Output:***

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 422ms/step

Sample Predictions (Iris): [[3.3180893e-03 8.5267818e-01 1.4400370e-01]

[9.9687386e-01 3.1254927e-03 6.0617316e-07]

[8.8813994e-07 2.1274907e-03 9.9787164e-01]

[3.5194594e-03 8.0342835e-01 1.9305217e-01]

[2.2715572e-03 9.0141135e-01 9.6317157e-02]]

***Save Models***

*# 1. Save the model for Abalone (Regression)*

linear\_model\_abalone.save('best\_model\_abalone.keras')

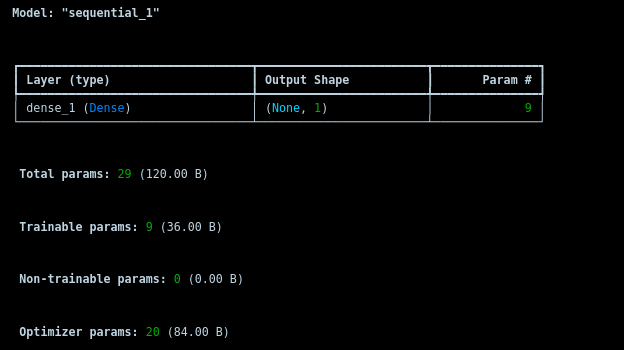
*# 2. Save the model for Iris (Classification)*

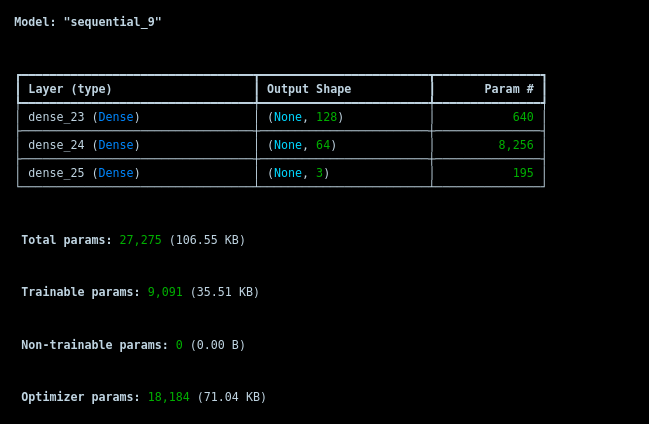
mlp\_model\_iris.save('best\_model\_iris.keras')

***20. Summary***

linear\_model\_abalone.summary()

mlp\_model\_iris.summary()

****

****

***Plotting***

import matplotlib.pyplot as plt

from tensorflow.keras.models import load\_model

***# 1. Load the saved model for Abalone (Regression)***

best\_model\_abalone = load\_model('best\_model\_abalone.keras')

***# 2. Predict on test data for Abalone***

predictions\_abalone = best\_model\_abalone.predict(X\_test\_abalone)

***# 3. Plot Predictions vs True Values for Abalone***

plt.figure(figsize=(10, 6))

plt.scatter(y\_test\_abalone, predictions\_abalone, alpha=0.6)

plt.plot([min(y\_test\_abalone), max(y\_test\_abalone)], [min(y\_test\_abalone), max(y\_test\_abalone)], color='r', linestyle='--')

plt.title('Abalone Dataset: Predicted vs True Values')

plt.xlabel('True Values')

plt.ylabel('Predicted Values')

plt.show()

***# 4. Load the saved model for Iris (Classification)***

best\_model\_iris = load\_model('best\_model\_iris.keras')

***# 5. Predict on test data for Iris***

predictions\_iris = best\_model\_iris.predict(X\_test\_iris)

***# 6. Plot Predicted Probabilities for Iris (3 classes)***

plt.figure(figsize=(10, 6))

plt.bar(range(len(predictions\_iris[0])), predictions\_iris[0], color='b', alpha=0.6, label='Prediction Probabilities')

plt.title('Iris Dataset: Predicted Class Probabilities (First Sample)')

plt.xlabel('Class Index')

plt.ylabel('Probability')

plt.xticks(range(3), ['Setosa', 'Versicolor', 'Virginica']) # Iris class names

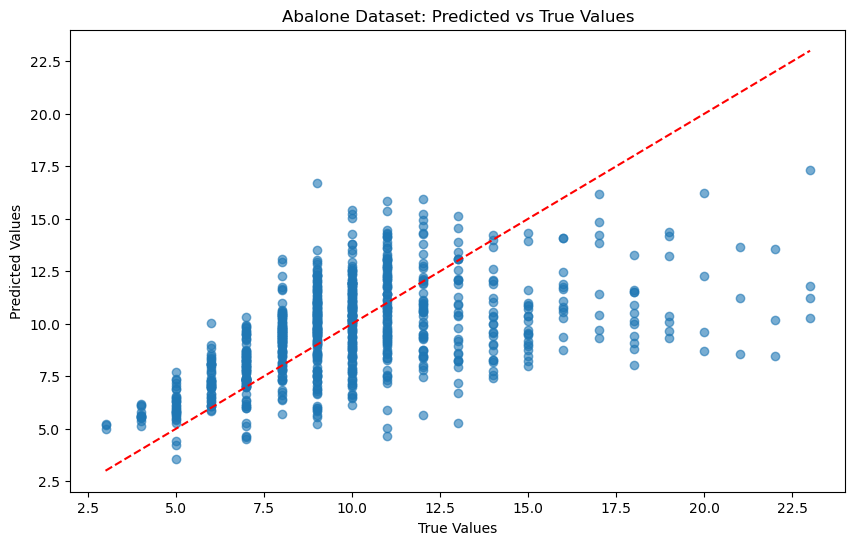
plt.show()

***# 7. Optionally: Convert predictions to class labels for Iris***

predicted\_classes\_iris = np.argmax(predictions\_iris, axis=1)

print("Predicted Classes (Iris):", predicted\_classes\_iris[:5])

***Output:***

****

