EXP NO:- 5

Date

Aim:- To implement a CNN for object detection in the given image.

Description:-

VGG16 is a convolution neural network (CNN) architecture which has been used to win ILSVR (Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. VGG16 is that instead of having large no. of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2.

program:-

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
from keras.applications import VGG16

TRAIN_DIR = "dataset/train"

TEST_DIR = "dataset/test"

IMAGE_SIZE = (128,128)

BATCH_SIZE = 2

train_datagen = ImageDataGenerator(
```

```
        rescale= 1./255,
        rotation_range=40
        width_shift_range=0.2,
        shear_range=0.2
        validation_split=0.4
)
test_datagen=ImageDataGenerator(
        rescale=1./255
        rotation_range=40
        shear_range=0.2
        horizontal_flip=True
)
train_generator=train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    class_mode='binary',
    subset="training",
)
test_generator=test_datagen.flow_from_directory(
    TEST_DIR,
    class_mode='binary',
)
weights_path="C:/Users/nik/Desktop/DL/Exp6/Exp6/
                vgg16.h5"
x=Flatten()
x=Dense(512, activation='relu')
x=Dropout(0.5)(x)
```

```
model.summary()
history=model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=10
)
ax=ax.ravel()
for i,met in enumerate(['accuracy','loss']):
  ax[i].plot(history.history[met])
  ax[i].plot(history.history['val'+met])
  ax[i].set_ylabel(met)
plt.show()
print("Test accuracy:",test_accuracy)
print("Test loss";test_loss)
```

Output :-

found 21 images belonging to 2 classes

found 13 " belonging to " "

found 11 " " " "

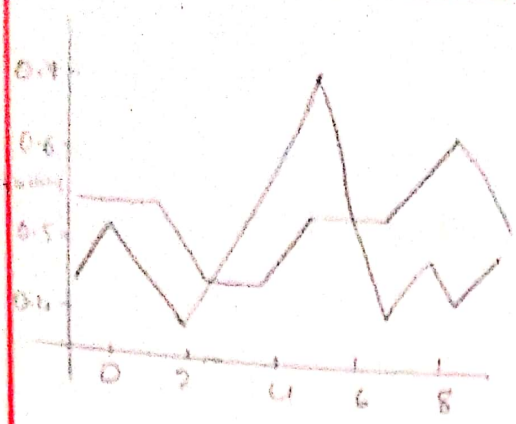
Model :"model"

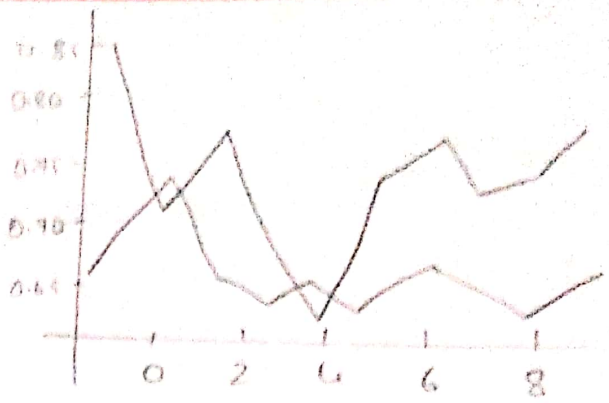
Layer(type)	Output shape	Param #

Total params: 18910017 (72.14 MB)

Trainable params: 18910017 (72.14 MB)

Non-trainable params: 0





Test accuracy : 0.6363632552642822

Test loss : 0.5794106721878052

11/3/24.

Expno -

Date:-

Aim:- To implement CNN for object detection in given image.

description:-

Object detection is a computer vision task that involves identifying and localizing objects within an image or video frame. Object detection within an image(or) video frame. object detection algorithm typically use deep learning techniques, such as CNN, to process input images and output bounding boxes around detected objects.

program:-

```
import keras
from keras.datacets import mnist
from keras.models import Sequential
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist
x_train = x_train.reshape(-1, 28, 28, 1)
y_train = keras.utils.to_categorically(y_train, 10)
y_test = keras.utils.to_categorically(y_test, 10)
```

```
model = Sequential([
    Conv2D(6, kernel_size=(5,5), activation='tanh',
           input_shape=(28,28,1)),
    AveragePooling2D(pool_size=(2,2), strides=(2,2)),
    Conv2D(16, kernel_size(5,5), activation='tanh'),
    AveragePooling2D(pool_size=(2,2), strides=(2,2)),
    flatten(),
Dense(120, activation='tanh'),
Dense(84, activation='tanh'),
Dense(10, activation='softmax')])
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
score = model.evaluate(x_test, y_test, verbose=0)
print(f'Test loss score: {score[0]}')
print(f'Test accuracy score: {score[1]}')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

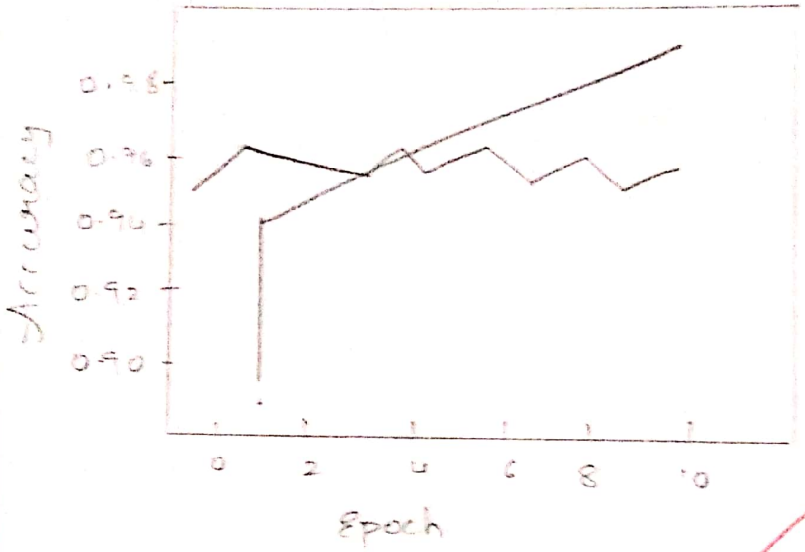
Output:-

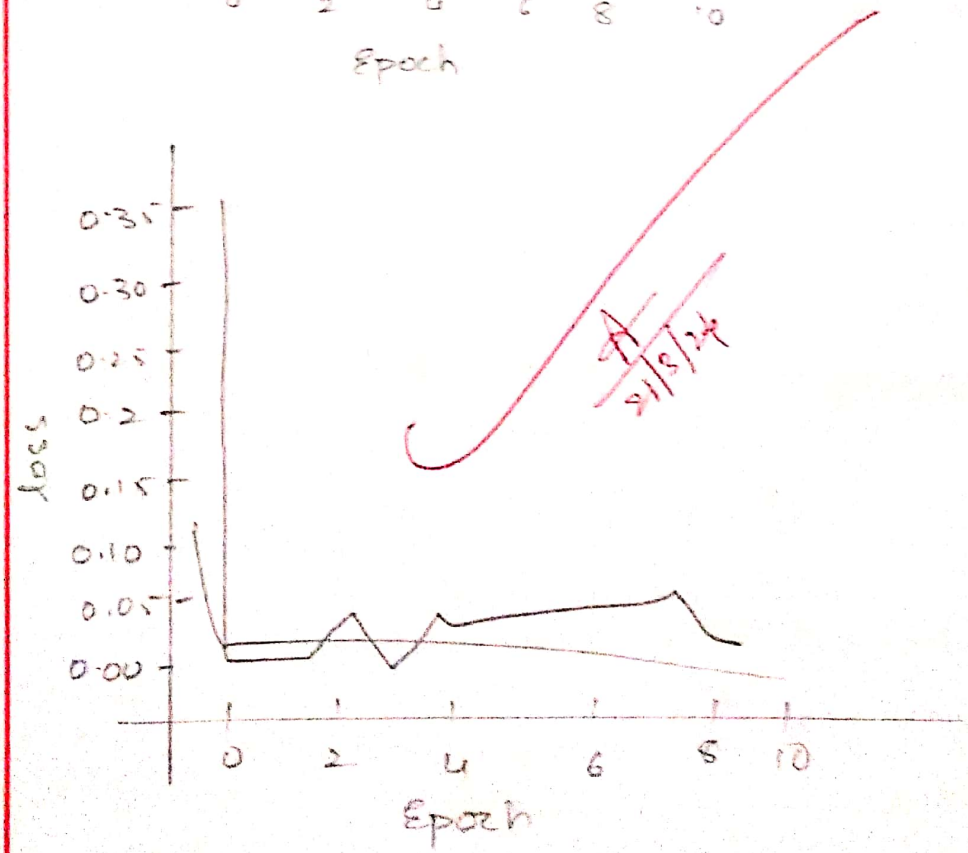
Epoch 1/12

:

Test loss score : 0.039396535605192184

Test accuracy score: 0.9868000149726868





Exp NO:- 8 Date:-

Aim:- Demonstrate working of dropout and early stopping on spotify dataset

Description:-

To demonstrate the working of dropout and early stopping on the spotify dataset, you would typically use a machine learning model, such as neural network, to predict some outcome based on features in the dataset. Implement: Load the dataset, preprocess the data, Build the neural network model, compile the model, Train the model, Evaluate the model. The model is trained with the the early stopping callback, and then evaluated on the testing data.

program:-

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
```

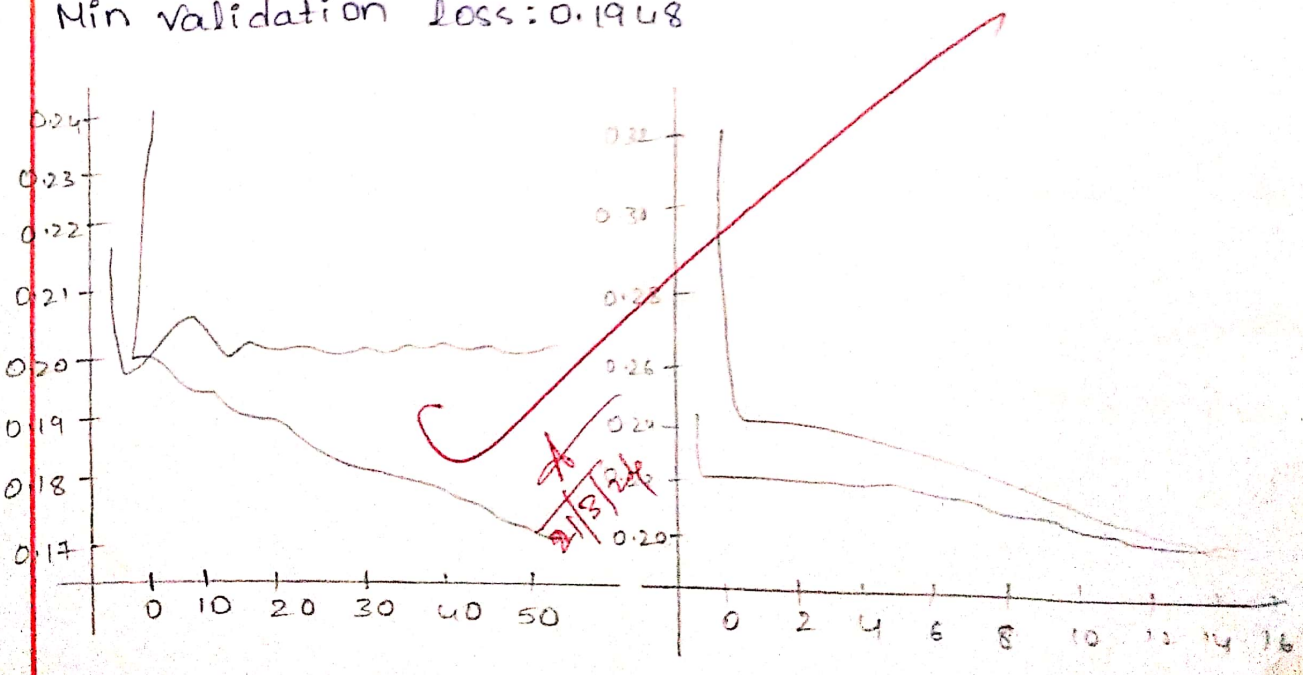
```
spotify = pd.DataFrame ("D:\\Desktop\\PDF's\\DL lab\\
                                    spotify")
X = spotify.dropna().copy()
y = X.pop('track_popularity')/100
artists = X['track_artist']
features_num = ['danceability', 'energy', 'key', 'loudness',
'mode', 'speechiness', 'acousticness', 'instrumentalness',
'liveness', 'valence', 'tempo', 'duration_ms']
features_cat = ['playlist_genre']
X = preprocessor.fit_transform(X)
X_train, X_temp, y_train, y_temp = train_test_split(
                X, y, test_size = 0.2, random_state = 42)
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape =
                                [X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
model.compile(optimizer='adam', loss='mae')
history = model.fit(
    X_train, y_train, validation_data = (X_valid, y_valid),
test_loss = model.evaluate(X_test, y_test)
```

```
print ("Test loss (MAE): {:.4f}".format (test_loss))
y_pred = model.predict (X_test)
mae = mean_absolute_error (y_test, y_pred)
history_df = pd.DataFrame (history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
print ("minimum validation loss: {:.0.4f}".format
                    (history_df['val_loss'].min()))
model_D_and_E = keras.Sequential([
    layers.Dropout (0.3),
    layers.Dense (64, activation='relu')
    layers.Dropout (0.3)
    layers.Dense (1)
model_D and_E compile (optimizer='adam', loss='mae')
 history = model_D and E.fit (
    X_train, y_train,
    validation_data = (X_valid, y_valid),
    batch_size = 512,
    epochs = 50,
callbacks = [early_stopping],
 verbose = 1
)
test_loss = model_D and E.evaluate (X_test, y_test)
print ("Test loss (MAE): {:.4f}".format (test_loss))
```

```
y_pred = model_D and E.predict(x_test)
print('Mean Absolute Error(MAE): {:.4f}'.format(mae))
history_df = pd.DataFrame(history.history)
history_df.loc[:,['loss','val_loss']].plot()
print('Minimum validation loss : {:0.4f}'.format
                    (history_df['val_loss'].min()))
```

Output :-

```
Epoch 1/50
 :
Mean Absolute Error(MAE): 0.1957
Min validation loss: 0.1925
 :
Mean Absolute Error(MAE): 0.1905
Min validation loss: 0.1948
```



EXP NO:- 9 Date:-

Aim:- To implement a RNN for predication time series data

description :-

A RNN is a type of artificial neural network designed to handle sequential data by maintaing a memory of past inputs. Unlike traditional feedforward neural networks, where data flows in one direction from input or output, RNNs have connections that form directed cycles, allowing them to exhibit temporal dynamic behaviour.

program :-

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import MinMax scaler.
import matplotlib.pyplot as plt.
weather_data = pd.read_csv ("C:/users/downloads/
                            weather_features.csv)
selected_features = ['temp', 'temp_min', 'temp_max',
    'pressure', 'humidity', 'wind_speed', 'rain_1h',
    'rain_3h', 'snow_3h']
x = weather_data[selected_features].values
```

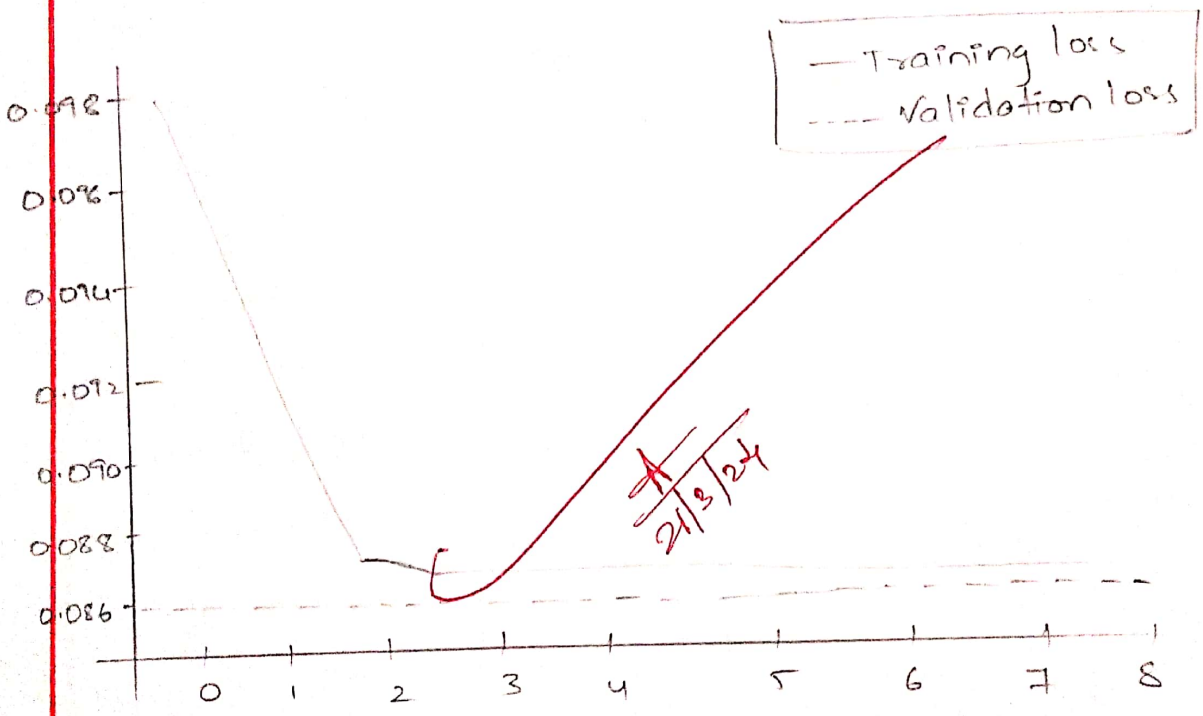
```
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
seq_length = 10
x_train = []
y_train = []
for i in range(len(x_scaled) - seq_length):
    x_train.append(x_scaled[i : i + seq_length])
    y_train.append(x_scaled[i + seq_length])
x_train, y_train = np.array(x_train), np.array(y_train)

model = Sequential([
  SimpleRNN(32, input_shape = (x_train.shape[1],
. Dropout(0.2),
 SimpleRNN(32),
Dropout(0.2),
Dense(1)
])
model.compile(optimizer = 'adam', loss = 'mean_squared_
                                          error')
train_loss = model.evaluate(x_train, y_train)
val_loss = model.evaluate(x_val, y_val)
print("Training loss:", train_loss)
print("validation loss:", val_loss)
plt.plot(history.history['loss'], label = 'Training loss')
plt.plot(history.history['val_loss'], label = 'validation
                                          loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

output :-

Training loss: 0.08654955029487761

validation loss : 0.08652949333190918



21/3/24

Exp NO:- 10 Date:-

Aim:- To implement a long short term memory (LSTM) for predicating time series data

Description:-

Long short term memory networks are a type of RNN architecture designed to address the vanishing gradient problem associated with traditional RNNs. LSTMs were introduced by Hochretier & schmidhuber in 1997 and have since become one of the most widely used architecture for sequential data processing tasks.

Program:-

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt.
weather_data= pd.read_csv('C:/users/nikhitha/downloads
                /weather_features.csv')
selected_features =['temp', 'temp_min', 'temp_max',
'pressure', 'humidity', 'wind_speed', 'rain_1h', 'rain_3h',
    'snow_3h', 'clouds_all']
```

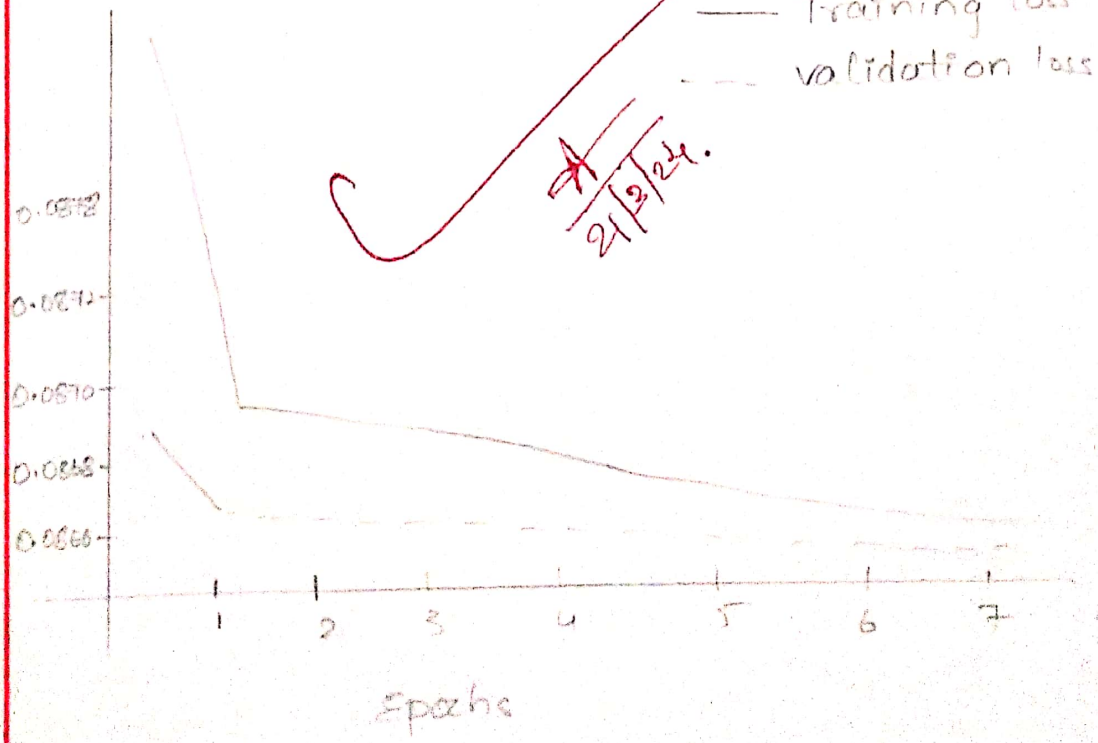
```
X = weather_data[selected_features].values
Scaler = MinMaxScaler()
X_scaled = Scaler.fit_transform(X)
seq_length = 10
X_train = []
y_train = []
for i in range(len(X_scaled) - seq_length):
    X_train.append(X_scaled[i:i+seq_length])
    y_train.append(X_scaled[i+seq_length])
X_train, y_train = np.array(X_train), np.array(y_train)

model = Sequential([
LSTM(32, input_shape=(X_train.shape[1], X_train.
        shape[2]), return_sequences=True),
Dropout(0.2),
LSTM(32),
Dropout(0.2),
Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_
                                        error')

train_loss = model.evaluate(X_train, y_train)
val_loss = model.evaluate(X_val, y_val)
print("Training loss:", train_loss)
print("Validation loss:", val_loss)
```

```
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='validation loss')

plt.xlabel('Epochs')
plt.ylabel('loss')
plt.title('Training & validation data')
plt.legend()
plt.show()
```

output:-

Training loss : 0.08660843223333359

validation loss : 0.08659161627292633



21/2/24.