KALASALINGAM
ACADEMY OF RESEARCH AND EDUCATION
UNIVERSITY
Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A" Grade

Department of Computer Science and Engineering

213CSE3301

DEEP LEARNING

LAB RECORD

2023-2024

Student Name : ....

Register Number : ....

Section :

TABLE OF CONTENTS

12	Using AlexNet	
13	RNN-LSTM	
14	GAN Implementation	

KALASALINGAM
Academy of Research and Education
UNIVERSITY
Estd.U/S 3 of UGC Act 1956. Accredited by NAAC with "A" Grade
www.kalasalingam.ac.in | 1800 425 7884 | 1800 425 9395

Anand Nagar, Krishnankoil - 626 126
Srivilliputtur (via), Virudhunagar (Dt)
Tamil Nadu, INDIA.
Ph : 04563 289012 / 22 / 32 / 42 / 52 / 72
e-mail : admissions@klu.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Bonafide record of work done by ____

of____________________ in ____

during even/odd semester in academic year ______________________

Staff In-charge Head of the Department

Submitted to the practical Examination held at Kalasalingam University, Krishnankoil on

----

Reg. No :

9	9	2	1	0	0	4	1	4	7	0

Internal Examiner External Examiner

EXPERIMENT EVALUATION SUMMARY

Name:Madhav kumar RegNo: 99210041470

Class: Faculty: Dr.T.Manikumar

S.No	Date	Experiment	Marks	Faculty Signature
1		Understanding the Perceptron		
2		Understanding the Perceptron Using Diabetes Dataset		
3		Building a Single-layer neural network		
4		Building a Multi-layer neural network		
5		Experiment with Activation Functions		
6		Experiment with Vehicle type recognition		
7		Diabetic Retinopathy		
8		Experimenting with different optimizers		
9		Improving generalization with regularization		
10		Adding dropout to prevent overfitting		
11		Image Augmentation		
12		Using AlexNet		
13		RNN-LSTM		

14		GAN Implementation		

Ex-1 Understanding the Perceptron

Aim: Implementing a python program for understanding the perceptron using the Iris plants dataset.

Algorithm:

1. Import the specific libraries and also load the Iris dataset.
2. In the Iris Plants dataset we have 3 classes. We are considering 2 classes from it namely Versicolor and Setosa.
3. Next step is to plot the data for two of the four variables. (Assign some colors to the data points that can be differentiated.
4. We need to split the data into training and testing so we can validate our results.
5. The next step is to initialize the random weights and assign the bias value as 1.
6. Also assign or define the hyperparameters. Here hyperparameters are learning rate and epochs. Here epochs denote the iteration number over the training set.
7. Now we can start the training our perceptron with a for loop.
8. In this for loop we use simple step function as If the output is greater than 0.5, we predict as 1, else 0.
9. In this step we are computing MSE and we are updating the weights and bias. And we are determining the validation accuracy.
10. At last, we will plot the training loss and validation accuracy.

Program:

```
# Import the libraries and dataset import numpy as

np   from sklearn.model_selection import

train_test_split import matplotlib.pyplot as plt

# We will be using the Iris Plants

Database from sklearn.datasets import

load_iris

SEED = 2017

# The first two classes (Iris-Setosa and Iris-Versicolour) are linear

separable iris = load_iris() idxs = np.where(iris.target<2) X =

iris.data[idxs] y = iris.target[idxs]

# Let's plot the data for two of the four variables

plt.scatter(X[y==0][:,0],X[y==0][:,2], color='green', label='Iris-Setosa')
```

```
plt.scatter(X[y==1][:,0],X[y==1][:,2],         color='red',         label='Iris-
Versicolour')


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=SEED)
# Next, we initialize the weights and the bias for the
perceptron weights =
np.random.normal(size=X_train.shape[1]) bias = 1
# Before training, we need to define the
hyperparameters learning_rate = 0.1 n_epochs
= 15 np.zeros(weights.shape)
# Now, we can start training our perceptron with a
for loop del_w = np.zeros(weights.shape)
hist_loss = [] hist_accuracy = [] for i in
range(n_epochs):
   # We apply a simple step function, if the output is > 0.5 we predict
1, else 0     output = np.where((X_train.dot(weights)+bias)>0.5, 1, 0)
print(output)
    # Compute MSE     error = np.mean((y_trainoutput)**2)
print("Error: ", error)     # Update weights and bias
weights-= learning_rate * np.dot((outputy_train), X_train)
bias += learning_rate * np.sum(np.dot((output-y_train),
X_train))     print("Weights:", weights)     print("bias:",
bias)     # Calculate MSE     loss = np.mean((output -
y_train) ** 2)     hist_loss.append(loss)     output_val =
np.where(X_val.dot(weights)>0.5, 1, 0)     accuracy =
np.mean(np.where(y_val==output_val, 1, 0))
hist_accuracy.append(accuracy)
# We've saved the training loss and validation accuracy so that we can
plot them fig = plt.figure(figsize=(8, 4)) a = fig.add_subplot(1,2,1)
imgplot = plt.plot(hist_loss) plt.xlabel('epochs')
a.set_title('Training
loss')
```

a=fig.add_subplot(1,

2,2) imgplot =

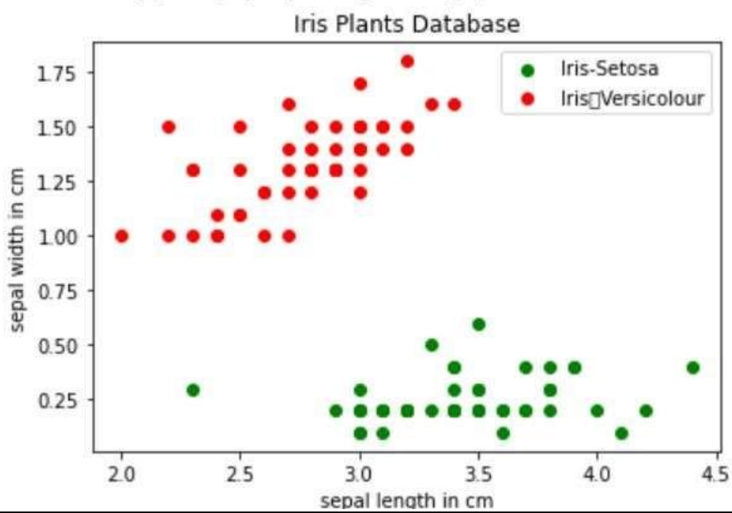
plt.plot(hist_accuracy

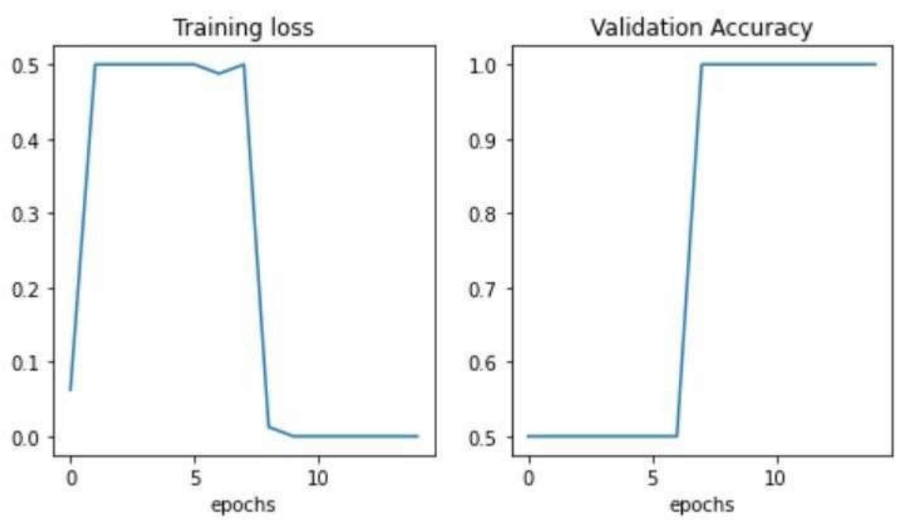
) plt.xlabel('epochs')

a.set_title('Validation Accuracy')

plt.show()

Output:





Result:

Ex-2 Understanding the Perceptron using Diabetes Dataset

Aim: Implementing a python program for understanding the perceptron using the Diabetes dataset.

Algorithm:

1. Import the specific libraries and also load the diabetics dataset.
2. In the Iris Plants dataset we have 3 classes. We are considering 2 classes from it namely yes or no class.
3. Next step is to plot the data for two of the four variables. (Assign some colors to the data points that can be differentiated.
4. We need to split the data into training and testing so we can validate our results.
5. The next step is to initialize the random weights and assign the bias value as 1.
6. Also assign or define the hyperparameters. Here hyperparameters are learning rate and epochs. Here epochs denote the iteration number over the training set.
7. Now we can start the training our perceptron with a for loop.
8. In this for loop we use simple step function as If the output is greater than 0.5, we predict as 1, else 0.
9. In this step we are computing MSE and we are updating the weights and bias. And we are determining the validation accuracy.
10. At last, we will plot the training loss and validation accuracy.

Program:

```
import numpy as np from sklearn import datasets from sklearn.model_selection
import train_test_split from sklearn.linear_model import Perceptron from
sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Step 1: Load the diabetes dataset
diabetes = datasets.load_diabetes()
X, y = diabetes.data, diabetes.target


# Step 2: Preprocess the
data X = X[:, np.newaxis,
2] y = (y > 140).astype(int)
```

```
# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Step 4: Create and train the perceptron perceptron =
Perceptron(max_iter=1000, random_state=42)
perceptron.fit(X_train, y_train)

# Step 5: Evaluate the perceptron
y_pred = perceptron.predict(X_test)

accuracy = accuracy_score(y_test,
y_pred) precision =
precision_score(y_test, y_pred) recall =
recall_score(y_test, y_pred) f1 =
f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

OUTPUT:

Accuracy: 0.6629213483146067

Precision: 0.6470588235294118

Recall: 0.55

F1 Score: 0.5945945945945946

Result:

Ex-3 Building a single-layer neural network

Aim: Implementation of a single-layer neural network using a python program.

Algorithm:

1. Import the necessary libraries, such as NumPy and Scikit-Learn.
2. Prepare the dataset for training and testing. This can include loading the data, splitting it into training and testing sets, and preprocessing the data as needed.
3. Define the architecture of the neural network. This includes the number of input and output neurons and the activation function to be used.
4. Initialize the weights and biases of the network randomly.
5. Define the forward propagation step. This includes calculating the dot product of the input data and the weights, adding the biases, and passing the result through the activation function.
6. Define the backward propagation step. This includes calculating the error, adjusting the weights and biases, and repeating this process for a number of epochs.
7. Use the trained network to make predictions on new data.
8. Finally, evaluate the performance of the network using metrics such as accuracy or mean squared error

Program:

```
# Import libraries and dataset import

numpy as np  from

sklearn.model_selection import

train_test_split import matplotlib.pyplot as

plt

# We will be using make_circles from

scikitlearn from sklearn.datasets import

make_circles

SEED = 2017

X, y = make_circles(n_samples=400, factor=.3, noise=.05,

random_state=2017) outer = y == 0 inner = y plt.title("Two Circles")

plt.plot(X[outer, 0], X[outer, 1], "ro") plt.plot(X[inner, 0], X[inner, 1],

"bo") plt.show() == 1

X = X+1
```

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=SEED) def sigmoid(x):   return 1 / (1 + np.exp(-x)) n_hidden = 50 # number of hidden units n_epochs = 1000 learning_rate = 1 # Initialise weights weights_hidden = np.random.normal(0.0, size=(X_train.shape[1], n_hidden)) weights_output = np.random.normal(0.0, size=(n_hidden))

hist_loss = []
hist_accuracy
= []
print(weights_
hidden)
print(weights_
output)

Run the single-layer neural network and output the statistics

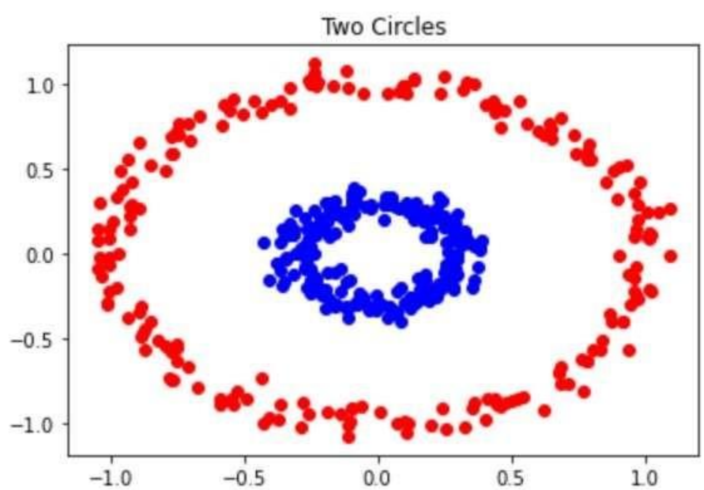
for e in range(n_epochs):
del_w_hidden =
np.zeros(weights_hidden.shape)
del_w_output =
np.zeros(weights_output.shape)

Loop through training data in batches of 1   for x_, y_ in zip(X_train, y_train):   # Forward computations   hidden_input = np.dot(x_, weights_hidden)   hidden_output = sigmoid(hidden_input)   output = sigmoid(np.dot(hidden_output, weights_output))

Backward computations   error = y_ - output   output_error = error * output * (1 - output)   hidden_error = np.dot(output_error, weights_output) * hidden_output * (1 - hidden_output)   del_w_output += output_error * hidden_output   del_w_hidden += hidden_error * x_[:, None]

```
    # Update weights      weights_hidden += learning_rate *
del_w_hidden / X_train.shape[0]      weights_output +=
learning_rate * del_w_output / X_train.shape[0]


    # Print stats (validation loss and
accuracy)      if e % 100 == 0:
hidden_output = sigmoid(np.dot(X_val,
weights_hidden))          out =
sigmoid(np.dot(hidden_output,
weights_output))          loss = np.mean((out -
y_val) ** 2)
        # Final prediction is based on a
threshold of 0.5          predictions = out >
0.5          accuracy = np.mean(predictions
== y_val)          print("Epoch: ",
'{:>4}'.format(e),
            "; Validation loss: ", '{:>6}'.format(loss.round(4)),
            "; Validation accuracy: ", '{:>6}'.format(accuracy.round(4)))
```

Output:



```
[[ 1.41581372  0.30855533 -0.8286774   0.74709373 -0.75963231  0.91704885
  -0.48495246 -0.57719441 -1.20896818 -1.70586972  0.0808446  -0.77745088
  -0.19259674  0.58443765 -0.22142047 -2.0896038  -1.53054408  1.10733381
  -0.42364537  0.49150686  0.25480185  0.75049925 -1.33832894  0.09060373
  -1.25736457 -0.38720421 -0.89152878  0.68514566  1.42506614  0.91491817
  -0.53204396 -1.46323081  1.28273973  1.70652197 -0.6210264  -0.20222603
   1.9963953   0.98509854 -0.90295226 -0.84067831 -0.78573054  0.96945578
   0.56616831 -0.38477712 -0.38741088 -0.49245588 -0.54497559 -0.76882439
  -0.8366077   0.75120829]
 [ 0.4278623   0.63510106 -1.43815054 -2.38940829  2.31206136 -1.52797071
  -0.7157763  -1.41321883  1.15097609  0.05966685 -1.39672623 -0.94216272
   0.83699277 -1.20327686  1.78814056 -1.01663469  0.67465652 -1.30155999
   0.36316766  2.7735055  -0.1750078  -0.05698066 -0.62223596 -0.20296104
   0.65097765 -0.03755962  1.04325509 -1.31261736 -1.05026247 -0.89180348
   0.41868647  0.46701954 -0.32783136 -0.87465801 -0.57567589 -0.19351514
  -0.98123078  1.56535402 -1.38543232  0.05637089  0.37151442 -0.09902364
  -0.6690398   0.73661872  0.8353953  -0.15306034  0.93446976  0.20623259
  -1.04797761 -0.38989319]]
[-1.46723602  2.07427738  0.14974924 -0.62524334 -0.22265344 -0.99273752
  0.36758329 -0.4332854   1.23580824  1.14726009 -0.99254906 -0.14468125
 -0.92648489  0.56625814  1.37419703  0.07928502 -0.66485609  0.74515938
  2.24707644 -0.84992104 -0.92020449  0.7256309  -0.00498305 -0.47895869
  0.25113205 -0.01462018 -0.53323325 -0.76871911 -0.7214322   0.25179481
  1.20416429 -1.2483621   2.0772874  -0.7569394  -0.16157457 -1.2922913
  0.5915649  -0.36964394  1.57888113  0.34552302 -0.60195869 -0.65268296
  0.38297428 -0.52851238  1.81118583 -0.26517826 -2.32389165  1.16327978
 -0.94309564 -0.10307125]
Epoch:   999 ; Validation loss:  0.2433 ; Validation accuracy:  0.5875
```

Result:

Ex-4 Building a Multi-Layer Neural Network

Aim: Implementation of a multi-layer neural network using a python program.

Algorithm:

1. Import the necessary libraries, such as NumPy and Scikit-Learn.
2. Prepare the dataset for training and testing. This can include loading the data, splitting it into training and testing sets, and preprocessing the data as needed.
3. Define the architecture of the neural network. This includes the number of input and output neurons, the number of hidden layers, and the number of neurons in each hidden layer.
4. Initialize the weights and biases of the network randomly.
5. Define the forward propagation step. This includes calculating the dot product of the input data and the weights, adding the biases, and passing the result through the activation function.

   Then repeat the process for the next layers.
6. Define the backward propagation step. This includes calculating the error, adjusting the weights and biases, and repeating this process for a number of epochs.
7. Use the trained network to make predictions on new data.
8. Finally, evaluate the performance of the network using metrics such as accuracy or mean squared error.

```
# We start by import the libraries

import numpy as np import pandas as pd from
sklearn.model_selection import train_test_split

from keras.models import Sequential from keras.layers
import Dense from keras.callbacks import EarlyStopping,
ModelCheckpoint from tensorflow.keras.optimizers import
Adam from sklearn.preprocessing import StandardScaler

SEED = 2017
```

```
# Data can be downloaded at https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv
```

```
# Load the wine data set

data = pd.read_csv('/content/winequality-red.csv', sep=';')
y = data['quality']
X = data.drop(['quality'], axis=1)
```

```
data
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
...	...	...	...	...	...	...	...	...	...
1594	6.20.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	
1595	5.90.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	
1596	6.30.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	
1597	5.90.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	
1598	6.00.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	

1599 rows × 12 columns

```
# Split data for training and testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=SEED)
```

```
# Print average quality and first rows of training set

print('Average quality training set: {:.4f}'.format(y_train.mean()))
X_train.head()
```

```
Average quality training set: 5.6231
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
1140	7.3	0.40	0.30	1.7	0.080	33.0	79.0	0.99690	3.41
920	9.6	0.41	0.37	2.3	0.091	10.0	23.0	0.99786	3.24
1198	7.7	0.26	0.26	2.0	0.052	19.0	77.0	0.99510	3.15
423	10.5	0.24	0.47	2.1	0.066	6.0	24.0	0.99780	3.15

```
# An important next step is to normalize the input data

scaler = StandardScaler().fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train))
X_test = pd.DataFrame(scaler.transform(X_test))
```

```
X_train
```

	0	1	2	3	4	5	6	
0	-0.580122	-0.703061	0.135441	-0.580168	-0.157722	1.640595	0.967796	0.09916
1	0.766902	-0.648140	0.493456	-0.166836	0.060168	-0.555900	-0.700855	0.60801
2	-0.345857	-1.471949	-0.069140	-0.373502	-0.712349	0.303598	0.908201	-0.85493
3	1.293998	-1.581790	1.004907	-0.304613	-0.435036	-0.937899	-0.671057	0.57620
4	2.875287	-0.373537	1.260633	-0.235724	-0.335995	-0.364900	-0.343287	2.06035
...	...	...	...	...	...	...	...	.
1274	0.708336	-0.867823	1.260633	-0.373502	-0.118105	-0.937899	-0.611463	0.68221
1275	0.649770	-1.087505	1.618649	0.177608	-0.177530	-0.651400	-0.492273	0.47019
1276	-0.697254	1.905667	-1.143187	2.175381	-0.118105	-1.224398	-0.969031	0.72462
1277	-0.462989	-0.153855	0.442311	2.450936	-0.335995	0.112599	1.653134	0.57620
1278	-1.400050	0.120748	-0.887462	-0.235724	-0.514268	2.213594	0.133471	-0.84433

1279 rows × 11 columns

```
# Determine baseline predictions
# Predict the mean quality of the training data for each validation input

print('MSE:', np.mean((y_test - ([y_train.mean()] * y_test.shape[0])) ** 2))
```

```
MSE: 0.5939855630834537
```

```
print('MSE:', np.mean((y_test - ([y_train.mean()] * y_test.shape[0])) ** 2))
```

```
MSE: 0.5939855630834537
```

```
# Now, let's build our neural network by defining the network architecture

model = Sequential()

# First hidden layer with 100 hidden units model.add(Dense(200,
input_dim=X_train.shape[1], activation='relu'))

# Second hidden layer with 50 hidden units
model.add(Dense(25, activation='relu'))

# Output layer model.add(Dense(1,
activation='linear'))

# Set optimizer
opt = Adam()

# Compile model model.compile(loss='mse', optimizer=opt,
metrics=['accuracy'])
```

```
# Define the callback for early stopping and saving the best model

callbacks = [
            EarlyStopping(monitor='val_accuracy', patience=30, verbose=2),
            ModelCheckpoint('checkpoints/multi_layer_best_model.h5', monitor='val_accuracy', save_best_only=True, verbose=0)
]
```

```
# Run the model with a batch size of 64, 5,000 epochs, and a validation split of 20%
```

```
batch_size = 64
n_epochs = 5000
```

```
X_train.values
```

```
array([[-0.58012193, -0.70306079,  0.13544056, ...,  0.6144467 ,
        -0.06322337, -0.88660767],
       [ 0.76690205, -0.6481402 ,  0.4934563 , ..., -0.47633666,
        -0.57072976,  0.05040374],
       [-0.34585689, -1.47194898, -0.06913987, ..., -1.05381021,
         0.72623102,  0.42520831],
       ...,
       [-0.69725445,  1.90566702, -1.14318708, ...,  0.55028298,
        -0.79628815, -0.23069968],
       [-0.46298941, -0.15385493,  0.44231119, ...,  0.22946434,
         0.78262062,  0.05040374],       [-1.40004958,
0.120748  , -0.88746156, ...,  1.32024771,
         0.55706222,  0.70631173]])
```

```
X_test.values
```

```
array([[ 0.53263701,  0.56011268, -0.32486539, ..., -0.21968175,
        -0.45795056, -0.79290653],
       [ 1.41113092,  0.77979502, -0.27372029, ..., -0.21968175,
         1.7976334 , -0.5118031 ],       [-0.87295323,
0.34043034, -1.09204198, ...,  0.6144467 ,
        -0.40156096,  0.33150716],
       ...,
       [ 1.05973336, -0.53829903,  0.64689161, ..., -0.60466412,        -
0.11961296, -0.41810196],
       [ 0.59120327,  0.69741414, -0.06913987, ..., -0.41217294,        -
1.36018414, -0.79290653],
       [-0.63868819, -1.0325843 , -0.32486539, ...,  0.35779179,        -
0.68350895, -0.32440082]])
```

```
model.fit(X_train.values, y_train, batch_size=64, epochs=n_epochs, validation_split=0.2,
verbose=2,               validation_data=(X_test.values, y_test),
callbacks=callbacks)
```

```
20/20 - 0s - loss: 2.2933 - accuracy: 0.0000e+00 - val_loss: 2.3013 - val_accuracy: 0.0000e+00 - 94ms/epoch - 5ms/step
Epoch 5/5000
20/20 - 0s - loss: 1.9380 - accuracy: 0.0000e+00 - val_loss: 1.9920 - val_accuracy: 0.0000e+00 - 100ms/epoch - 5ms/step
Epoch 6/5000
20/20 - 0s - loss: 1.7241 - accuracy: 0.0000e+00 - val_loss: 1.8194 - val_accuracy: 0.0000e+00 - 90ms/epoch - 5ms/step
Epoch 7/5000
20/20 - 0s - loss: 1.5705 - accuracy: 0.0000e+00 - val_loss: 1.7033 - val_accuracy: 0.0000e+00 - 92ms/epoch - 5ms/step
Epoch 8/5000
20/20 - 0s - loss: 1.4393 - accuracy: 0.0000e+00 - val_loss: 1.5786 - val_accuracy: 0.0000e+00 - 95ms/epoch - 5ms/step
Epoch 9/5000
20/20 - 0s - loss: 1.3227 - accuracy: 0.0000e+00 - val_loss: 1.4757 - val_accuracy: 0.0000e+00 - 91ms/epoch - 5ms/step
Epoch 10/5000
20/20 - 0s - loss: 1.2187 - accuracy: 0.0000e+00 - val_loss: 1.3989 - val_accuracy: 0.0000e+00 - 92ms/epoch - 5ms/step
Epoch 11/5000
20/20 - 0s - loss: 1.1391 - accuracy: 0.0000e+00 - val_loss: 1.3259 - val_accuracy: 0.0000e+00 - 88ms/epoch - 4ms/step
Epoch 12/5000
20/20 - 0s - loss: 1.0486 - accuracy: 0.0000e+00 - val_loss: 1.2309 - val_accuracy: 0.0000e+00 - 102ms/epoch - 5ms/step
Epoch 13/5000
20/20 - 0s - loss: 0.9843 - accuracy: 0.0000e+00 - val_loss: 1.1654 - val_accuracy: 0.0000e+00 - 91ms/epoch - 5ms/step
Epoch 14/5000
20/20 - 0s - loss: 0.9154 - accuracy: 0.0000e+00 - val_loss: 1.1113 - val_accuracy: 0.0000e+00 - 78ms/epoch - 4ms/step
Epoch 15/5000
20/20 - 0s - loss: 0.8541 - accuracy: 0.0000e+00 - val_loss: 1.0513 - val_accuracy: 0.0000e+00 - 80ms/epoch - 4ms/step
Epoch 16/5000
20/20 - 0s - loss: 0.8093 - accuracy: 0.0000e+00 - val_loss: 0.9868 - val_accuracy: 0.0000e+00 - 91ms/epoch - 5ms/step
Epoch 17/5000
20/20 - 0s - loss: 0.7573 - accuracy: 0.0000e+00 - val_loss: 0.9497 - val_accuracy: 0.0000e+00 - 95ms/epoch - 5ms/step
Epoch 18/5000
20/20 - 0s - loss: 0.7109 - accuracy: 0.0000e+00 - val_loss: 0.9013 - val_accuracy: 0.0000e+00 - 117ms/epoch - 6ms/step
Epoch 19/5000
20/20 - 0s - loss: 0.6730 - accuracy: 0.0000e+00 - val_loss: 0.8322 - val_accuracy: 0.0000e+00 - 78ms/epoch - 4ms/step
Epoch 20/5000
20/20 - 0s - loss: 0.6373 - accuracy: 0.0000e+00 - val_loss: 0.8031 - val_accuracy: 0.0000e+00 - 70ms/epoch - 4ms/step
Epoch 21/5000
20/20 - 0s - loss: 0.6068 - accuracy: 0.0000e+00 - val_loss: 0.7639 - val_accuracy: 0.0000e+00 - 67ms/epoch - 3ms/step
Epoch 22/5000
20/20 - 0s - loss: 0.5678 - accuracy: 0.0000e+00 - val_loss: 0.7318 - val_accuracy: 0.0000e+00 - 82ms/epoch - 4ms/step
Epoch 23/5000
20/20 - 0s - loss: 0.5516 - accuracy: 0.0000e+00 - val_loss: 0.6811 - val_accuracy: 0.0000e+00 - 66ms/epoch - 3ms/step
Epoch 24/5000
20/20 - 0s - loss: 0.5207 - accuracy: 0.0000e+00 - val_loss: 0.6858 - val_accuracy: 0.0000e+00 - 77ms/epoch - 4ms/step
Epoch 25/5000
20/20 - 0s - loss: 0.5062 - accuracy: 0.0000e+00 - val_loss: 0.6580 - val_accuracy: 0.0000e+00 - 81ms/epoch - 4ms/step
Epoch 26/5000
```

```
20/20 - 0s - loss: 0.4876 - accuracy: 0.0000e+00 - val_loss: 0.6170 - val_accuracy: 0.0000e+00 - 72ms/epoch - 4ms/step
Epoch 27/5000
20/20 - 0s - loss: 0.4691 - accuracy: 0.0000e+00 - val_loss: 0.5861 - val_accuracy: 0.0000e+00 - 77ms/epoch - 4ms/step
Epoch 28/5000
20/20 - 0s - loss: 0.4539 - accuracy: 0.0000e+00 - val_loss: 0.5784 - val_accuracy: 0.0000e+00 - 78ms/epoch - 4ms/step
Epoch 29/5000 20/20 - 0s - loss: 0.4413 - accuracy: 0.0000e+00 - val_loss: 0.5594 - val_accuracy: 0.0000e+00 -
82ms/epoch - 4ms/step
Epoch 30/5000
20/20 - 0s - loss: 0.4264 - accuracy: 0.0000e+00 - val_loss: 0.5416 - val_accuracy: 0.0000e+00 - 100ms/epoch - 5ms/step
Epoch 31/5000
20/20 - 0s - loss: 0.4193 - accuracy: 0.0000e+00 - val_loss: 0.5463 - val_accuracy: 0.0000e+00 - 75ms/epoch - 4ms/step
Epoch 31: early stopping
<keras.src.callbacks.History at 0x7a870254c1c0>
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
dense_6 (Dense)             (None, 200)               2400
dense_7 (Dense)             (None, 25)                5025
dense_8 (Dense)             (None, 1)                 26
=================================================================
Total params: 7451 (29.11 KB)
Trainable params: 7451 (29.11 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# We can now print the performance on the test set after loading the optimal weights:

best_model = model best_model.load_weights('checkpoints/multi_layer_best_model.h5')
best_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

# Evaluate on test set score = best_model.evaluate(X_test.values, y_test, verbose=0) print('Test accuracy:
%.2f%%' % (score[0]))

# Test accuracy: 65.62%
# Benchmark accuracy on dataset 62.4%
```

```
Test accuracy: 15.30%
```

Result

Ex-5 Experiment with Activation Functions

Aim: Implement a python program for getting started with the activation function.

Algorithm:

1. Import the necessary libraries for your algorithms, such as NumPy and Matplotlib for data manipulation and visualization.
2. Initialize your dataset and choose an activation function to use, such as sigmoid, ReLU, or tanh.
3. Define the function for the chosen activation function using NumPy.
4. Apply the activation function to your dataset using the function you just defined.
5. Visualize the activated data using Matplotlib to observe the effect of the activation function on the dataset.
6. Repeat s 2-5 for any other activation functions you wish to test on your dataset.
7. Depending on the context, you may want to use the activation function as part of a neural network. For that you can use pre-built library such as TensorFlow or PyTorch.

Program:

```
# Import the libraries as follows import
numpy as np  import pandas as pd  from
sklearn.model_selection import
train_test_split import matplotlib.pyplot
as plt from keras.models import
Sequential from keras.layers import
Dense from tensorflow.keras.utils import
to_categorical from keras.callbacks
import Callback from keras.datasets
import mnist
SEED = 2022
# Load the MNIST dataset
# Need Internet Connection to download dataset
(X_train, y_train), (X_val, y_val) = mnist.load_data()
# Show an example of each label and print the count per label
# Plot first image of each label unique_labels =
set(y_train) plt.figure(figsize=(12, 12)) i = 1 for label in
unique_labels:     image =
X_train[y_train.tolist().index(label)]     plt.subplot(10, 10,
i)     plt.axis('off')     plt.title("{0}: ({1})".format(label,
y_train.tolist().count(label)))     i += 1
    _ = plt.imshow(image,
cmap='gray') plt.show()
print(X_val) print(y_val)
# Preprocess the data
# Normalize data
X_train = X_train.astype('float32')/255.
X_val = X_val.astype('float32')/255.
X_val
# One-Hot-Encode
labels n_classes = 10
y_train =
```

```
to_categorical(y_train,
n_classes) y_val =
to_categorical(y_val,
n_classes) print(y_train)
# Flatten data - we treat the image as a sequential array of values
X_train = np.reshape(X_train, (60000, 784))
X_val = np.reshape(X_val, (10000, 784))
X_train
# Define the model with the sigmoid activation function
model_sigmoid = Sequential()  model_sigmoid.add(Dense(700,
input_dim=784, activation='sigmoid'))
model_sigmoid.add(Dense(700, activation='sigmoid'))
model_sigmoid.add(Dense(700, activation='sigmoid'))
model_sigmoid.add(Dense(700, activation='sigmoid'))
model_sigmoid.add(Dense(700, activation='sigmoid'))
model_sigmoid.add(Dense(350, activation='sigmoid'))
model_sigmoid.add(Dense(100, activation='sigmoid'))
model_sigmoid.add(Dense(10, activation='softmax'))
# Compile model with SGD  model_sigmoid.compile(loss='categorical_crossentropy',
optimizer='sgd', metrics=['accuracy'])
# Define the model with the ReLU activation function
model_relu                    =                    Sequential()
model_relu.add(Dense(700,             input_dim=784,
activation='relu'))           model_relu.add(Dense(700,
activation='relu'))           model_relu.add(Dense(700,
activation='relu'))           model_relu.add(Dense(700,
activation='relu'))           model_relu.add(Dense(700,
activation='relu'))           model_relu.add(Dense(350,
activation='relu'))           model_relu.add(Dense(100,
activation='relu'))             model_relu.add(Dense(10,
activation='softmax'))
# Compile model with SGD  model_relu.compile(loss='categorical_crossentropy',
optimizer='sgd', metrics=['accuracy'])  # Create a callback function to store the loss
values per batch class history_loss(Callback):     def on_train_begin(self, logs={}):
```

self.losses = [] def
on_batch_end(self, batch,
logs={}):

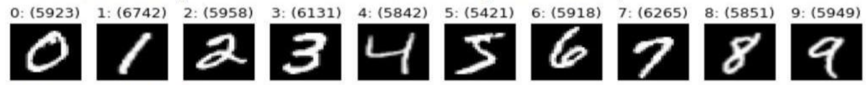
batch_loss =
logs.get('loss')
self.losses.append(batch _loss) n_epochs = 10 batch_size = 256 validation_split = 0.2 history_sigmoid = history_loss() model_sigmoid.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size, callbacks=[history_sigmoid], validation_split=validation_split, verbose=2) history_relu = history_loss() model_relu.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size, callbacks=[history_relu], validation_split=validation_split, verbose=2) np.arange(len(history_sigmoid.losses)) print(history_sigmoid.losses) plt.plot(np.arange(len(history_sigmoid.losses)), history_sigmoid.losses, label='sigmoid') plt.plot(np.arange(len(history_relu.losses)), history_relu.losses, label='relu') plt.title('Losses for sigmoid and ReLU model') plt.xlabel('number of batches') plt.ylabel('loss') plt.legend(loc=1) plt.show()

Losses for sigmoid and ReLU model

# Extract the maximum weights of each model per layer
w_sigmoid = [] w_relu = [] for i in
range(len(model_sigmoid.layers)):
w_sigmoid.append(max(model_sigmoid.layers[i].get_weights(
)[1]))
w_relu.append(max(model_relu.layers[i].get_weights()[1]))
print(w_sigmoid) print(w_relu)

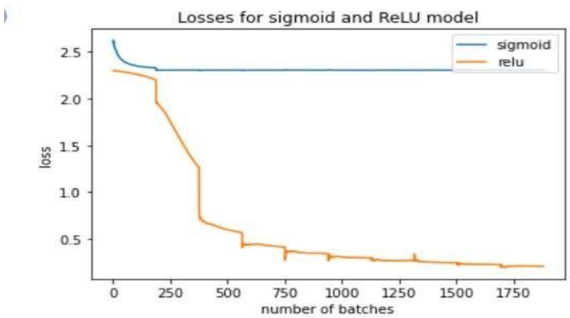
print(len(model_sigmoid.layers)) #
Plot the weights of both models fig,
ax = plt.subplots() index =
np.arange(len(model_sigmoid.layers
)) bar_width = 0.35 plt.bar(index, w_sigmoid, bar_width, label='sigmoid', color='b', alpha=0.4) plt.bar(index + bar_width, w_relu, bar_width, label='relu', color='r', alpha=0.4) plt.title('Maximum weights across layers for sigmoid and ReLU activation functions') plt.xlabel('layer number') plt.ylabel('maximum weight') plt.legend(loc=0) plt.xticks(index + bar_width / 2, np.arange(8)) plt.show() Output:

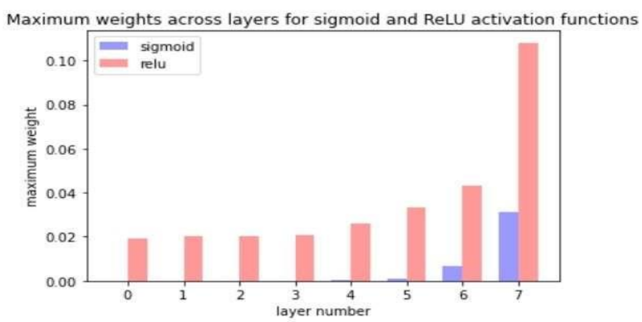
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```




```
[[[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]]
```

```
# Losses for sigmoid and ReLU model
```





6. Experiment with Vehicle Type Recognition

Aim: To implement Vehicle type recognition in python language.

Algorithm:

1. Gather a dataset of vehicle images, labeled with their corresponding vehicle types (e.g., car, truck, motorcycle).Split the dataset into training and testing sets.
2. Import required libraries, including TensorFlow or PyTorch for deep learning and OpenCV for image processing.
3. Resize all images to a common size (e.g., 224x224 pixels) to ensure consistent input dimensions for the CNN.
    a. Normalize pixel values to a common range (e.g., [0, 1] or [0, 255]).
    b. Optionally, apply data augmentation techniques (e.g., random rotation, flipping) to increase model robustness.

4. Create a CNN model consisting of convolutional layers (Conv2D), pooling layers (MaxPooling2D), and fully connected layers (Dense).
    a. Adjust the number of layers and filters based on the complexity of your task.
    b. Use activation functions like ReLU and appropriate kernel sizes.
    c. Add a softmax output layer with as many neurons as there are vehicle classes, and use categorical cross-entropy as the loss function.
5. Compile the CNN model by specifying the optimizer (e.g., Adam), loss function (categorical cross-entropy), and evaluation metric (accuracy).
6. Train the model using the training dataset.
    a. Specify the number of epochs and batch size.
    b. Monitor training progress and loss convergence.
7. Assess the model's performance using the test dataset.Calculate accuracy and other relevant metrics to evaluate its effectiveness.
8. Use the trained CNN model to make predictions on new vehicle images.
9. Visualize predictions, confusion matrices, and model performance metrics for better understanding.
10. Experiment with different architectures, hyperparameters, and data augmentation techniques to improve model performance.
11. If needed, deploy the trained model in a real-world application for vehicle type recognition.

Vehicle Type Recognition

by Johann

Data Info

This is a vehicle image classi cation dataset containing images of four different types of vehicles: Car, Truck, Bus, and Motorcycle. The dataset is curated to help learners to develop and evaluate image classi cation models for identifying various vehicle types from images.

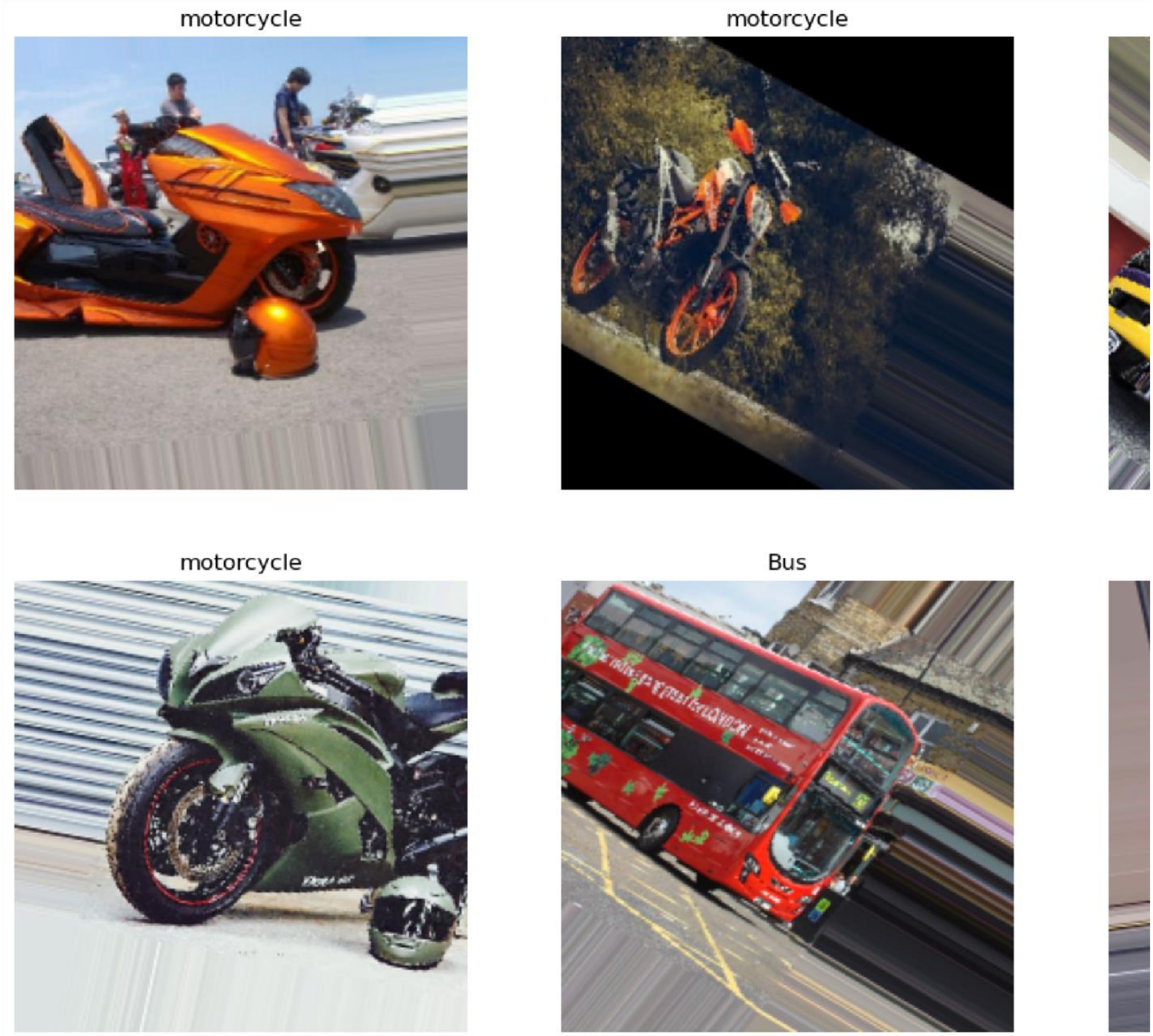
The image collection is made into a separate Kaggle notebook. https://www.kaggle.com/code/kaggleashwin/datasetcollection-for-vehicle-type-recognition

Data

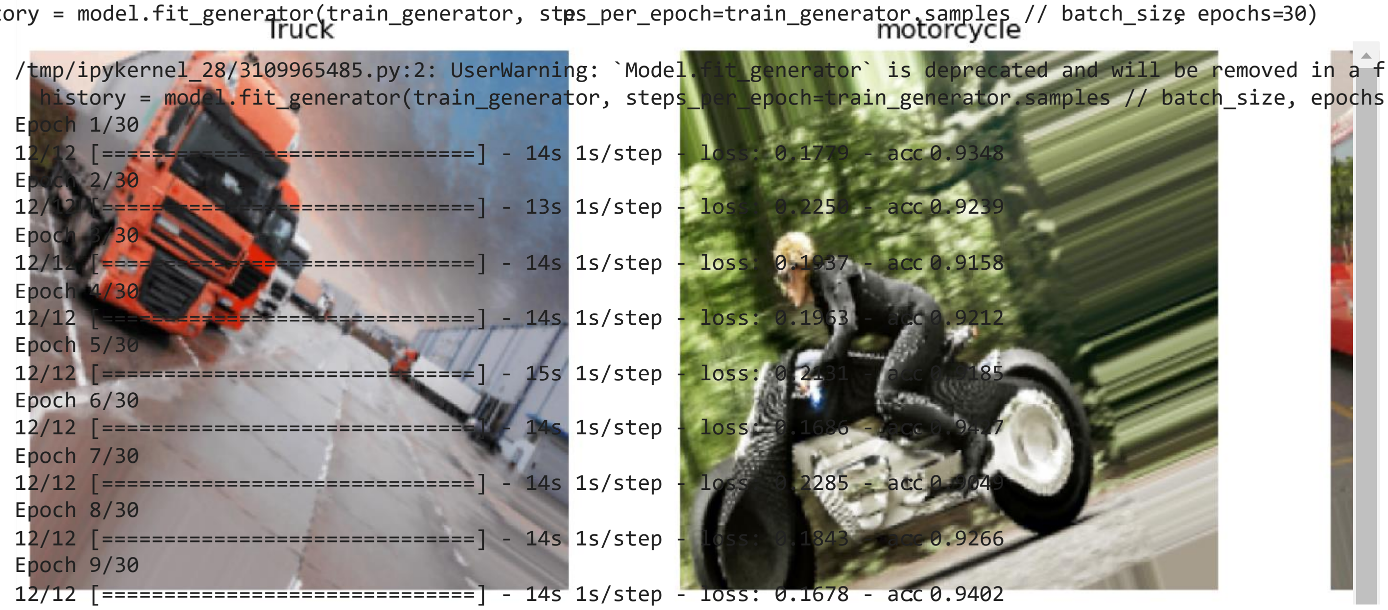
```
import os import
numpy as np
from keras.preprocessing.image import
ImageDataGenerator from keras.applications import
VGG16 from keras import models, layers, optimizers

# Set the path to the dataset folder
dataset_path = '/kaggle/input/vehicle-type-recognition/Dataset'

# Set the batch size and image
size batch_size = 32 image_size =
(224, 224)

# Create an instance of the ImageDataGenerator class for data
augmentation train_datagen = ImageDataGenerator(     rescale=1./255,
rotation_range=40,     width_shift_range=0.2,
height_shift_range=0.2,     shear_range=0.2,     zoom_range=0.2,
horizontal_flip=True,     fill_mode='nearest')

# Create a generator for the training data
train_generator =
train_datagen.flow_from_directory(
dataset_path,     target_size=image_size,
batch_size=batch_size,
class_mode='categorical')

# Load the VGG16 model with pre-trained weights base_model = VGG16(weights='imagenet',
include_top=False, input_shape=(image_size[0], image_size[1], 3))

# Freeze the layers of the base model
for layer in base_model.layers:
layer.trainable = False

# Create a new model by adding custom layers on top of the base
model model = models.Sequential() model.add(base_model)
model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))

# Compile the model model.compile(loss='categorical_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-4), metrics=['acc'])
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and
<1.23.0   warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to load
lib caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undef   warnings.warn(f"unable to load
libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system
plugi caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so:
undefined sym   warnings.warn(f"file system plugins are not loaded: {e}")
Found 400 images belonging to 4 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_or
58889256/58889256 [==============================] - 0s 0us/step
/opt/conda/lib/python3.10/site-packages/keras/optimizers/legacy/rmsprop.py:143: UserWarning: The `lr`
argument i   super().__init__(name, **kwargs)
```

```
import matplotlib.pyplot as plt

# Get the class labels
class_labels = train_generator.class_indices
class_labels = dict((v, k) for k, v in class_labels.items())

# Display the images and their labels fig, ax
= plt.subplots(3, 3, figsize=(15, 15)) for i
in range(9):     x, y =
train_generator.next()     image = x[0]
label = y[0]     label = np.argmax(label)
label = class_labels[label]     ax[i//3,
i%3].imshow(image)     ax[i//3,
i%3].set_title(label)     ax[i//3,
i%3].axis('off') plt.show()
/opt/conda/lib/python3.10/site-
packages/PIL/Image.py:992: UserWarning:
Palette images with Transparency expresse
warnings.warn(
```



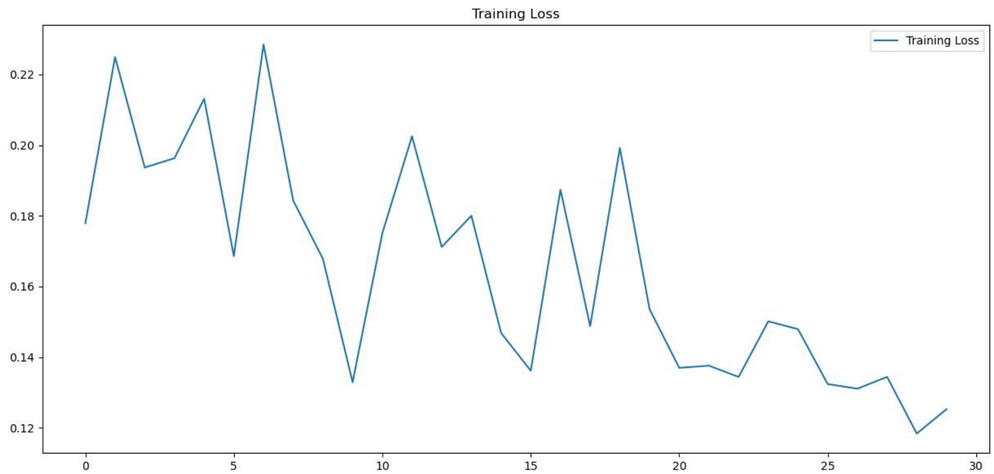

```
# Train the model
history = model.fit_generator(train_generator, steps_per_epoch=train_generator.samples // batch_size, epochs=30)
```




```
/tmp/ipykernel_28/3109965485.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a f
  history = model.fit_generator(train_generator, steps_per_epoch=train_generator.samples // batch_size, epochs
Epoch 1/30
12/12 [==============================] - 14s 1s/step - loss: 0.1779 - acc 0.9348
Epoch 2/30
12/12 [==============================] - 13s 1s/step - loss: 0.2250 - acc 0.9239
Epoch 3/30
12/12 [==============================] - 14s 1s/step - loss: 0.1937 - acc 0.9158
Epoch 4/30
12/12 [==============================] - 14s 1s/step - loss: 0.1963 - acc 0.9212
Epoch 5/30
12/12 [==============================] - 15s 1s/step - loss: 0.2131 - acc 0.9185
Epoch 6/30
12/12 [==============================] - 14s 1s/step - loss: 0.1686 - acc 0.9427
Epoch 7/30
12/12 [==============================] - 14s 1s/step - loss: 0.2285 - acc 0.9049
Epoch 8/30
12/12 [==============================] - 14s 1s/step - loss: 0.1843 - acc 0.9266
Epoch 9/30
12/12 [==============================] - 14s 1s/step - loss: 0.1678 - acc 0.9402
```

```
Epoch 10/30
12/12 [==============================] - 14s 1s/step - loss: 0.1329 - acc 0.9402
Epoch 11/30
12/12 [==============================] - 13s 1s/step - loss: 0.1750 - acc 0.9266
Epoch 12/30
12/12 [==============================] - 15s 1s/step - loss: 0.2025 - acc 0.9245
Epoch 13/30
12/12 [==============================] - 14s 1s/step - loss: 0.1712 - acc 0.9266
Epoch 14/30
12/12 [==============================] - 15s 1s/step - loss: 0.1800 - acc 0.9348
Epoch 15/30
12/12 [==============================] - 14s 1s/step - loss: 0.1469 - acc 0.9565
Epoch 16/30
12/12 [==============================] - 15s 1s/step - loss: 0.1361 - acc 0.9484
Epoch 17/30
12/12 [==============================] - 13s 1s/step - loss: 0.1874 - acc 0.9402
Epoch 18/30
12/12 [==============================] - 14s 1s/step - loss: 0.1488 - acc 0.9429
Epoch 19/30
12/12 [==============================] - 14s 1s/step - loss: 0.1992 - acc 0.9212
Epoch 20/30
12/12 [==============================] - 13s 1s/step - loss: 0.1536 - acc 0.9348
Epoch 21/30
12/12 [==============================] - 13s 1s/step - loss: 0.1370 - acc 0.9565
Epoch 22/30
12/12 [==============================] - 14s 1s/step - loss: 0.1376 - acc 0.9484
Epoch 23/30
12/12 [==============================] - 14s 1s/step - loss: 0.1344 - acc 0.9592
Epoch 24/30
12/12 [==============================] - 14s 1s/step - loss: 0.1501 - acc 0.9293
Epoch 25/30
12/12 [==============================] - 14s 1s/step - loss: 0.1479 - acc 0.9293
Epoch 26/30
12/12 [==============================] - 14s 1s/step - loss: 0.1324 - acc 0.9531
Epoch 27/30
12/12 [==============================] - 14s 1s/step - loss: 0.1311 - acc 0.9457
Epoch 28/30
```

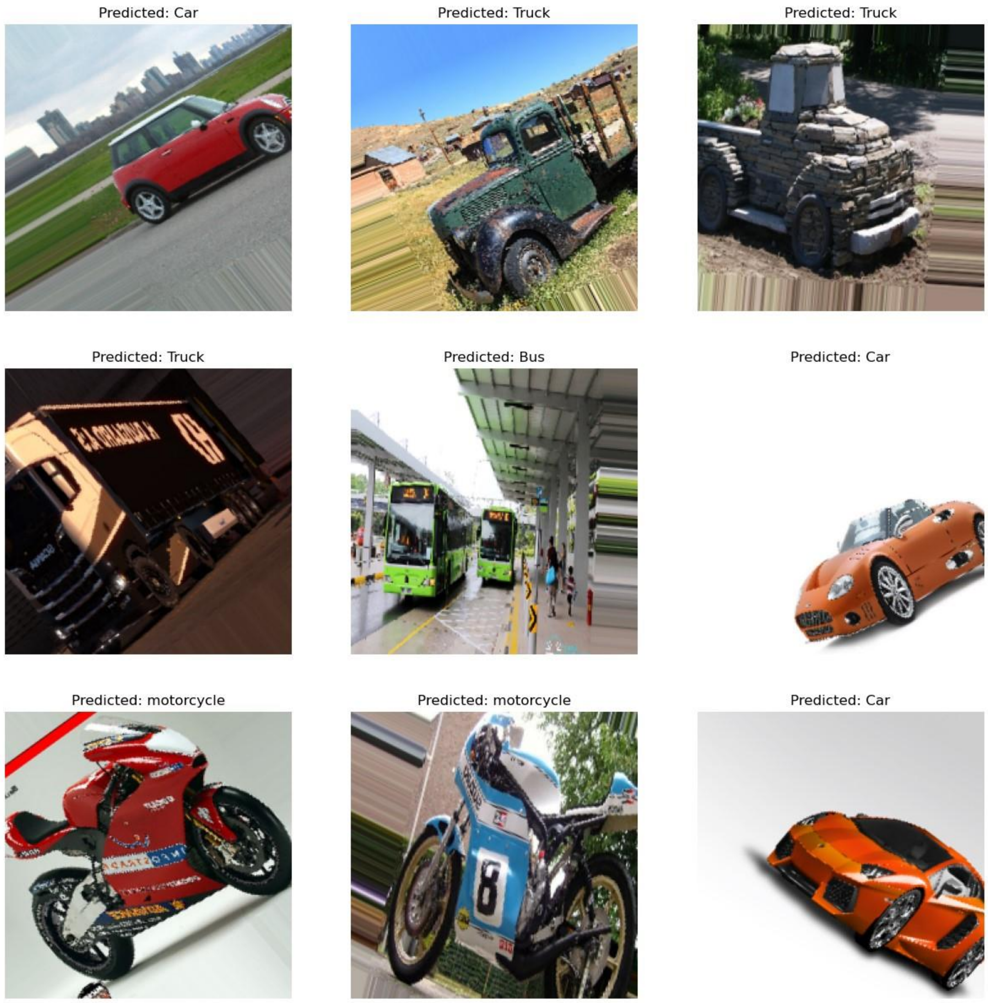
```
# Display the data

import matplotlib.pyplot as plt

acc = history.history['acc']
loss = history.history['loss']

epochs_range = range(len(acc))

plt.figure(figsize=(15, 15))
plt.subplot(2, 1, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.legend(loc='lower right')
plt.title('Training Accuracy')

plt.subplot(2, 1, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.legend(loc='upper right')
plt.title('Training Loss')
plt.show()
```




```
import matplotlib.pyplot as plt

# Get the class labels class_labels =
train_generator.class_indices class_labels = dict((v, k)
for k, v in class_labels.items())

# Display the images and their predicted
labels fig, ax = plt.subplots(3, 3,
figsize=(15, 15)) for i in range(9):     x, y
= train_generator.next()     image = x[0]
label = y[0]     # Get the predicted label
    pred = model.predict(np.expand_dims(image,
axis=0))     pred_label = np.argmax(pred)
pred_label = class_labels[pred_label]     ax[i//3,
i%3].imshow(image)
    ax[i//3, i%3].set_title(f'Predicted:
{pred_label}')     ax[i//3, i%3].axis('off')
plt.show()
```

```
     1/1 [==============================] - 0s 27ms/step
     1/1 [==============================] - 0s 22ms/step
     1/1 [==============================] - 0s 21ms/step
     1/1 [==============================] - 0s 22ms/step
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
```



7. Diabetic Retinopathy

Aim: To implement Diabetic Retinopathy in python language.

Algorithm:

1. Gather a dataset of retinal images, ideally labeled with diabetic retinopathy severity levels.Split the dataset into training and testing sets.
2. Import the necessary libraries, including deep learning frameworks like TensorFlow or PyTorch, and image processing libraries like OpenCV.
3. Preprocess the retinal images to enhance their quality and prepare them for analysis.Resize images to a common size (e.g., 224x224 pixels) for consistent input dimensions.
    a. Normalize pixel values to a common range (e.g., [0, 1] or [0, 255]).
    b. Augment the training data with techniques like rotation, flipping, and brightness adjustments to increase model robustness (optional).
4. Create a CNN model tailored for image classification.
    a. Use convolutional layers (Conv2D), pooling layers (MaxPooling2D), and fully connected layers (Dense).
    b. Adjust the number of layers and filters based on the complexity of the task.Employ activation functions like ReLU and kernel sizes suitable for image analysis.
    c. Add a softmax output layer with as many neurons as there are diabetic retinopathy severity levels (e.g., 0 to 4), and use categorical cross-entropy as the loss function.
5. Compile the CNN model by specifying the optimizer (e.g., Adam), loss function (categorical cross-entropy), and evaluation metric (e.g., accuracy).
6. Train the model using the training dataset.Specify the number of epochs and batch size.Monitor training progress, including loss convergence and validation performance.
7. Assess the model's performance using the test dataset.
    a. Calculate evaluation metrics such as accuracy, precision, recall, and F1-score.
    b. Examine the confusion matrix to understand the model's strengths and weaknesses.
8. Utilize the trained CNN model to make predictions on new retinal images. Interpret the predictions to determine the severity level of diabetic retinopathy.
9. Visualize model predictions, confusion matrices, and performance metrics for better insights and communication.

Program 7-Diabetic Retinopathy detection using CNN

```
# This Python 3 environment comes with many helpful analytics libraries inst alled
# It is defined by the kaggle/python Docker image: https://github.com/kaggle
/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory # For example, running this (by clicking run or pressing Shift+Enter) will l ist all files under the input directory
 import os for dirname, _, filenames in os.walk('/kaggle/input'):
   for filename in filenames:        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"  # You can also write temporary files to /kaggle/temp/, but they won't be sav ed outside of the current session
```

```
/kaggle/input/diabetic-retinopathy-detection/train.zip.003
/kaggle/input/diabetic-retinopathy-detection/test.zip.004
/kaggle/input/diabetic-retinopathy-detection/test.zip.005
/kaggle/input/diabetic-retinopathy-detection/train.zip.002
/kaggle/input/diabetic-retinopathy-detection/test.zip.006
/kaggle/input/diabetic-retinopathy-detection/test.zip.003
/kaggle/input/diabetic-retinopathy-detection/train.zip.005
/kaggle/input/diabetic-retinopathy-detection/train.zip.001
/kaggle/input/diabetic-retinopathy-detection/sampleSubmission.csv.zip
/kaggle/input/diabetic-retinopathy-detection/test.zip.007
/kaggle/input/diabetic-retinopathy-detection/trainLabels.csv.zip
/kaggle/input/diabetic-retinopathy-detection/test.zip.001
/kaggle/input/diabetic-retinopathy-detection/sample.zip
/kaggle/input/diabetic-retinopathy-detection/train.zip.004
/kaggle/input/diabetic-retinopathy-detection/test.zip.002
/kaggle/input/prepossessed-arrays-of-binary-data/Binary_images_data_128 .npz
/kaggle/input/prepossessed-arrays-of-binary-data/Binary_images_data_90. npz
/kaggle/input/prepossessed-arrays-of-binary-data/1000_Binary_images_dat a_264.npz
/kaggle/input/prepossessed-arrays-of-binary-data/1000_Binary Dataframe
/kaggle/input/prepossessed-arrays-of-binary-data/Binary_images_data_264
.npz
/kaggle/input/prepossessed-arrays-of-binary-data/1000_Binary_images_dat a_128.npz
/kaggle/input/prepossessed-arrays-of-binary-data/1000_Binary_images_dat a_90.npz
/kaggle/input/prepossessed-arrays-of-binary-data/Binary Dataframe
```

```
import numpy as np import pandas
as pd import cv2
from PIL import Image
```

```
import tensorflow as tf
from tensorflow.keras.optimizers import * from tensorflow.keras.losses
import * from tensorflow.keras.layers import * from
tensorflow.keras.models import * from tensorflow.keras.callbacks import
* from tensorflow.keras.preprocessing.image import * from
tensorflow.keras.utils import * from sklearn.metrics import * from
sklearn.model_selection import * import tensorflow.keras.backend as K

from tqdm import tqdm import matplotlib.pyplot
as plt import seaborn as sns from skimage.io
import * %config Completer.use_jedi = False
import time
from sklearn.metrics import confusion_matrix
 print("All modules have been imported")
```

All modules have been imported

In [3]:

```
info=pd.read_csv("../input/prepossessed-arrays-of-binary-data/1000_Binary D ataframe")
info=info.drop('Unnamed: 0',axis=1) info.head()
```

Out[3]:

	exists	eye_side	level	path	patient_id	level_cat
0	True	left	0	../input/diabetic-retinopathy-detection/10_lef...	10	[1. 0.]
1	True	right	0	../input/diabetic-retinopathy-detection/10_rig...	10	[1. 0.]
	exists	eye_side	level	path	patient_id	level_cat
2	True	left	0	../input/diabetic-retinopathy-detection/13_lef...	13	[1. 0.]
3	True	right	0	../input/diabetic-retinopathy-detection/13_rig...	13	[1. 0.]

4	True	left	0	../input/diabetic-retinopathy-detection/17_lef...	17	[1. 0.]

In [4]:

```
info.level.value_counts()
```

Out[4]:

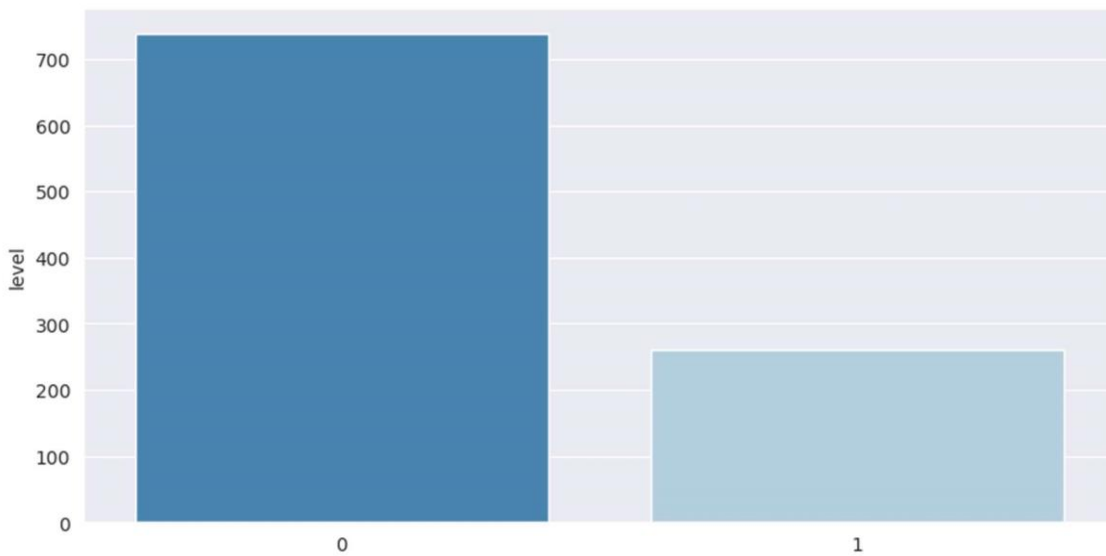
```
0        739
1        261
Name: level, dtype: int64
```

In [5]:

```
sns.set_style('darkgrid') fig, ax =
plt.subplots(figsize=(10,5))
sns.barplot(x=info.level.unique(),y=info.level.value_counts(),palette='Blue s_r',ax=ax)
```

Out[5]:

<AxesSubplot:ylabel='level'>



In [6]:

```
sizes = info['level'].values sns.distplot(sizes, kde=False)
```

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: UserWar ning:

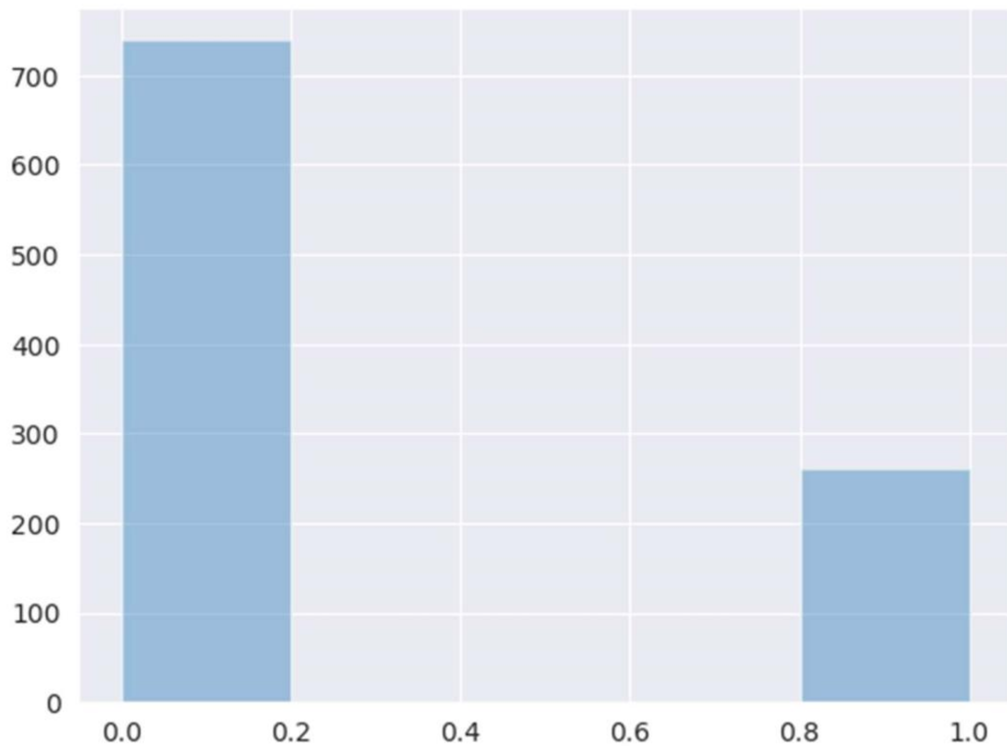
`distplot` is a deprecated function and will be removed in seaborn v0.1 4.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogra ms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

Out[6]:

<AxesSubplot:>



In [7]:

```
Binary_90 = np.load('../input/prepossessed-arrays-of-binary-data/1000_Binar y_images_data_90.npz')
X_90=Binary_90['a']
Binary_128 = np.load('../input/prepossessed-arrays-of-binary-data/1000_Bina ry_images_data_128.npz')
X_128=Binary_128['a']
Binary_264 = np.load('../input/prepossessed-arrays-of-binary-data/1000_Bina ry_images_data_264.npz')
X_264=Binary_264['a'] y=info['level'].values

print(X_90.shape)
print(X_128.shape) print(X_264.shape)
print(y.shape)
```

```
(1000, 24300)
(1000, 49152)
(1000, 209088)
(1000,)
```

In [8]:

```
print("Shape before reshaping X_90" +str(X_90.shape)) X_90=X_90.reshape(1000,90,90,3)
print("Shape after reshaping X_90" +str(X_90.shape)) print("\n\n")

print("Shape before reshaping X_128" +str(X_128.shape)) X_128=X_128.reshape(1000,128,128,3)
print("Shape after reshaping X_128" +str(X_128.shape)) print("\n\n")

print("Shape before reshaping X_264" +str(X_264.shape)) X_264=X_264.reshape(1000,264,264,3)
print("Shape after reshaping X_264" +str(X_264.shape))
```

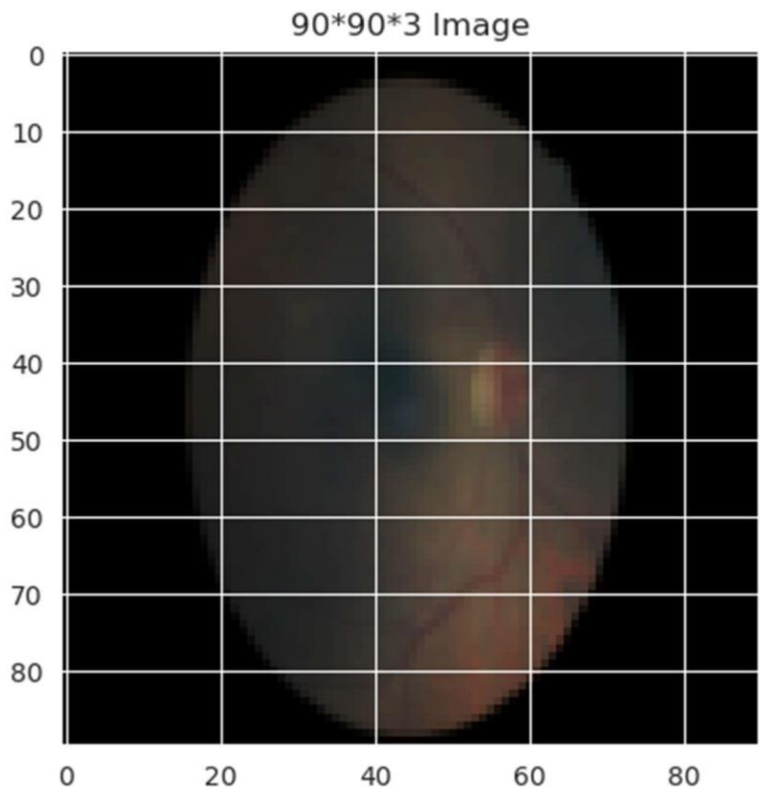
```
Shape before reshaping X_90(1000, 24300)
Shape after reshaping X_90(1000, 90, 90, 3)
```

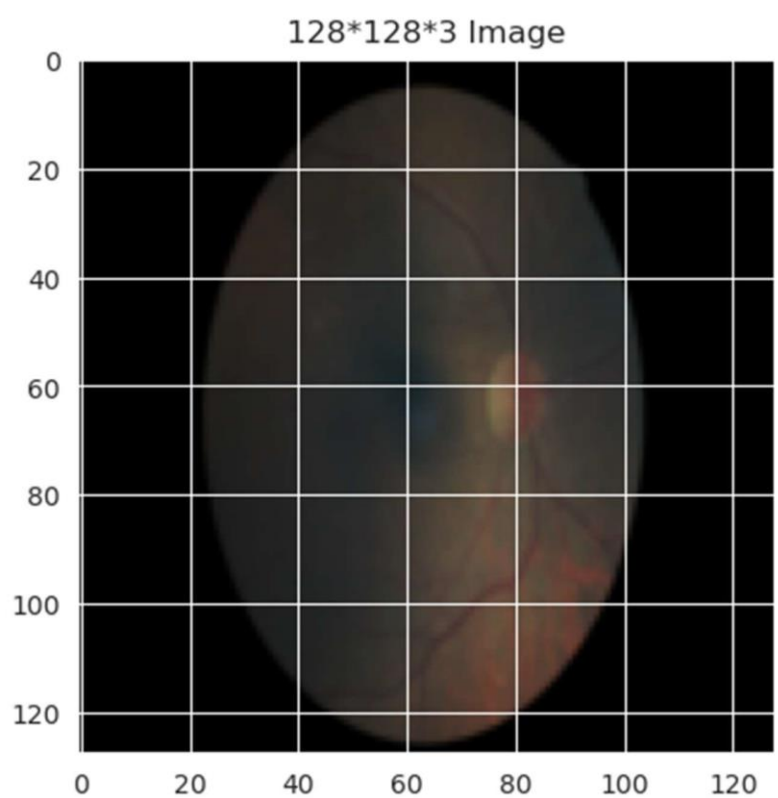
Shape before reshaping X_128(1000, 49152)
Shape after reshaping X_128(1000, 128, 128, 3)

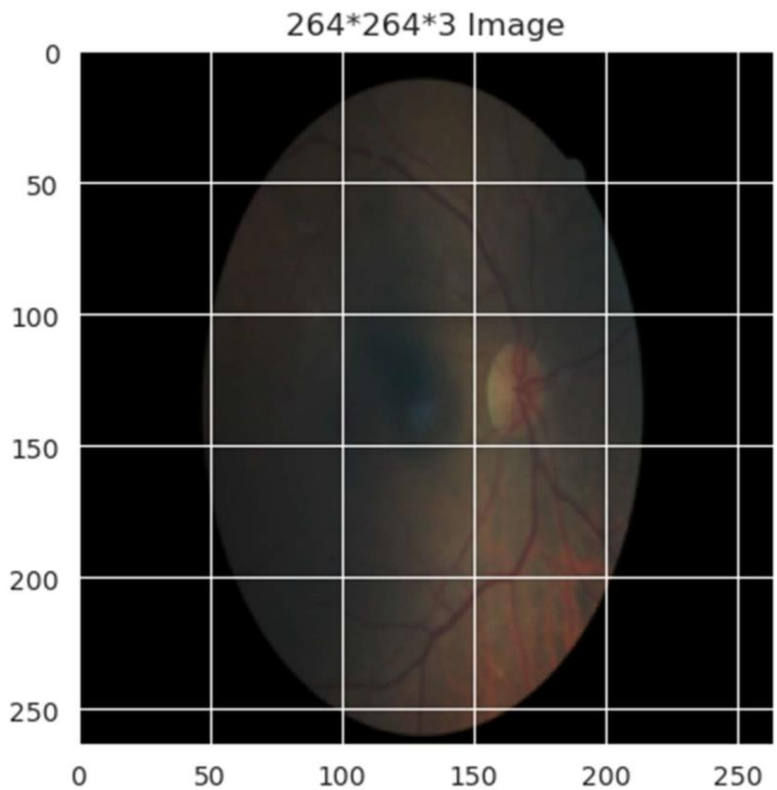
Shape before reshaping X_264(1000, 209088)
Shape after reshaping X_264(1000, 264, 264, 3)

In [9]:

```
plt.title("90*90*3 Image")
plt.imshow(X_90[1]) plt.show()
plt.title("128*128*3 Image")
plt.imshow(X_128[1]) plt.show()
plt.title("264*264*3 Image")
plt.imshow(X_264[1]) plt.show()
```


90*90*3 Image
0
10
20
30
40
50
60
70
80
0
20
40
60
80


128*128*3 Image
0
20
40
60
80
100
120
0
20
40
60
80
100
120



In [10]:

```
X=np.array(X_264)
Y=np.array(y) #
Y=to_categorical(Y,5)
x_train, x_test1, y_train, y_test1 = train_test_split(X, Y, test_size=0.4, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_test1, y_test1, test_size
=0.5, random_state=42)
print(len(x_train),len(x_val),len(x_test))
```

600 200 200

In [11]:

```
Y1=pd.DataFrame(Y)
Y1.value_counts()
```

Out[11]:

0   739 1   261
dtype: int64

In [12]:

```
# Create the CNN model model =
Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(264, 264, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(5, activation='softmax')
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met rics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_ data=(x_val, y_val))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test) print('Test accuracy:',
test_acc) Epoch 1/10
19/19 [==============================] - 34s 2s/step - loss: 1.2050 - a ccuracy: 0.6650 -
val_loss: 0.6270 - val_accuracy: 0.7300
Epoch 2/10
19/19 [==============================] - 33s 2s/step - loss: 0.6168 - a
ccuracy: 0.7267 - val_loss: 0.5939 - val_accuracy: 0.7300
Epoch 3/10
19/19 [==============================] - 32s 2s/step - loss: 0.5796 - a
ccuracy: 0.7267 - val_loss: 0.5840 - val_accuracy: 0.7300
Epoch 4/10
19/19 [==============================] - 33s 2s/step - loss: 0.5860 - a
ccuracy: 0.7267 - val_loss: 0.5901 - val_accuracy: 0.7300
Epoch 5/10
19/19 [==============================] - 33s 2s/step - loss: 0.5939 - a
ccuracy: 0.7183 - val_loss: 0.9219 - val_accuracy: 0.7300
Epoch 6/10
19/19 [==============================] - 32s 2s/step - loss: 0.6095 - a
ccuracy: 0.7267 - val_loss: 0.5889 - val_accuracy: 0.7300
Epoch 7/10
19/19 [==============================] - 33s 2s/step - loss: 0.5567 - a ccuracy: 0.7217 -
val_loss: 0.5907 - val_accuracy: 0.7200
Epoch 8/10
19/19 [==============================] - 32s 2s/step - loss: 0.5392 - a ccuracy: 0.7350 -
val_loss: 0.5872 - val_accuracy: 0.7100
Epoch 9/10
19/19 [==============================] - 32s 2s/step - loss: 0.5380 - a ccuracy: 0.7367 -
val_loss: 0.5794 - val_accuracy: 0.7250
Epoch 10/10
19/19 [==============================] - 33s 2s/step - loss: 0.5296 - a ccuracy: 0.7333 -
val_loss: 0.5875 - val_accuracy: 0.7150
7/7 [==============================] - 3s 381ms/step - loss: 0.5528 - a ccuracy: 0.7750
Test accuracy: 0.7749999761581421
```

In [13]:

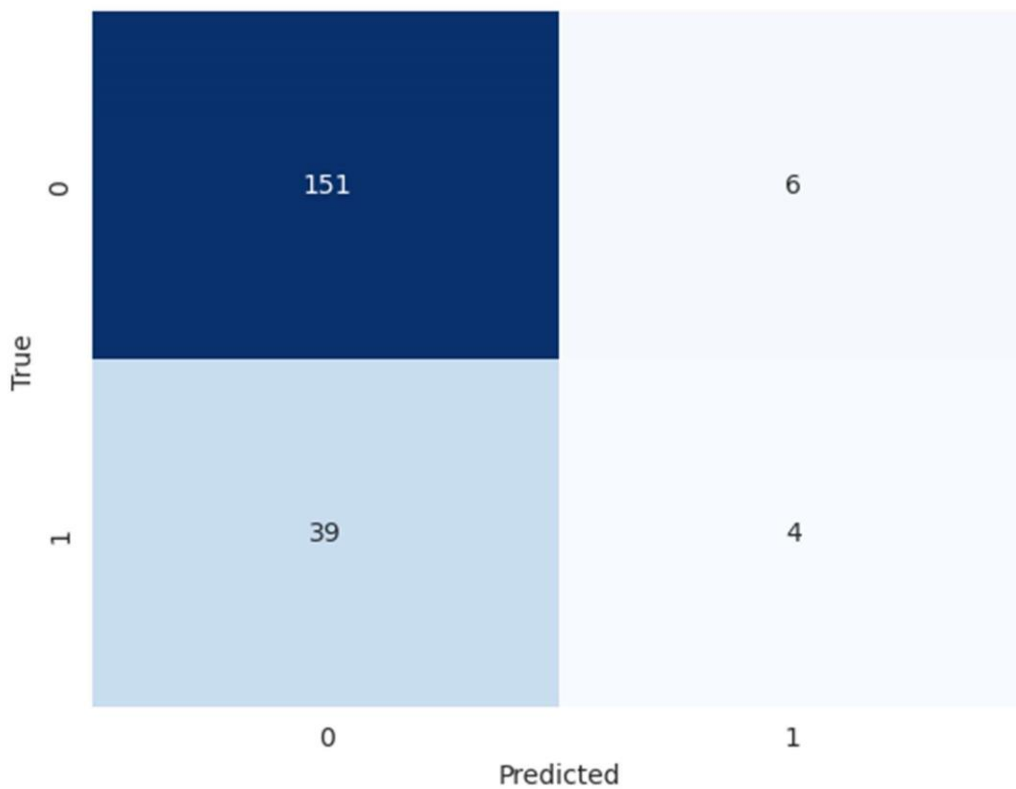
```
from sklearn.metrics import confusion_matrix

# Get the predicted class labels for the test set y_pred =
np.argmax(model.predict(x_test), axis=-1)
```

```
# Calculate the confusion matrix cm =
confusion_matrix(y_test, y_pred)

# Print the confusion matrix print("Confusion
Matrix:") print(cm)
```

```
7/7 [==============================] - 3s 357ms/step
Confusion Matrix:
[[151   6]
 [ 39   4]]
```

In [14]:

```
import seaborn as sns

# Visualize the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False) plt.xlabel("Predicted")
plt.ylabel("True") plt.show()
```



Ex-8 Experimenting with different optimizers

Aim: Implement a Python program for Experimenting with different optimizers Compare the results of the training for each optimizer and determine which optimizer performed the best.

Algorithm:

1: Import the necessary libraries, such as numpy, keras, and matplotlib.

2: Load the dataset to be used for training the model.

3: Define the model architecture.

4: Compile the model by specifying the loss function, metrics, and optimizer.

5: Create a list of optimizers to be tested, such as SGD, RMSprop, Adam, etc.

6: Create a loop that iterates over the list of optimizers. For each iteration: (i) set the optimizer for the model using the model.compile method (ii) fit the model on the dataset using the fit method (iii) store the results of the training, such as accuracy or loss.

7: Plot the results of the training for each optimizer, such as accuracy or loss, over the number of epochs.

8: Compare the results of the training for each optimizer and determine which optimizer performed .

Program: import numpy as np import

pandas as pd from

sklearn.model_selection import

train_test_split from keras.models

import

Sequential from keras.layers import Dense, Dropout from

keras.callbacks import EarlyStopping, ModelCheckpoint

from tensorflow.keras.optimizers import SGD, Adadelta, Adam, RMSprop, Adagrad, Nadam, Adamax

SEED = 2022

Data can be downloaded at https://archive.ics.uci.edu/ml/machine-

learningdatabases/winequality/winequality-red.csv data =

pd.read_csv('C:\\Users\\ifsrk\\Documents\\01 Deep Learning\\winequality-red.csv', sep=';') y

= data['quality']

X = data.drop(['quality'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=SEED)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,

test_size=0.2, random_state=SEED) SEED

print(np.any(np.isnan(X_test)))

print(np.any(np.isinf(X_test)))

print(np.any(np.isnan(X_train)))

print(np.any(np.isinf(X_train)))

print(np.any(np.isnan(y_test)))

print(np.any(np.isinf(y_test)))

```
print(np.any(np.isnan(y_train)))
print(np.any(np.isinf(y_train)))


def create_model(opt):      model = Sequential()
model.add(Dense(100,
input_dim=X_train.shape[1],
activation='relu'))     model.add(Dense(50,
activation='relu'))     model.add(Dense(25,
activation='relu'))     model.add(Dense(10,
activation='relu'))     model.add(Dense(1,
activation='linear'))     return model def
create_callbacks(opt):
   callbacks = [
   EarlyStopping(monitor='accuracy', patience=50, verbose=2),
   ModelCheckpoint('checkpoints/optimizers_best_' + opt + '.h5', monitor='accuracy',
save_best_only=True, verbose=1)
   ]
   return
callbacks
opts =
dict({
   'sgd': SGD(),
    'sgd-0001': SGD(learning_rate=0.0001, decay=0.00001),
    'adam': Adam(),
    'adadelta': Adadelta(),
    'rmsprop': RMSprop(),
    'rmsprop-0001': RMSprop(learning_rate=0.0001),
    'nadam': Nadam(),
    'adamax': Adamax()


}
)
X_train.values batch_size = 128 n_epochs = 1000
```

```
results = []

# Loop through the
optimizers for opt in opts:
model =
create_model(opt)
callbacks =
create_callbacks(opt)
model.compile(loss='mse'
, optimizer=opts[opt],
metrics=['accuracy'])

#   model.compile(loss='mse', optimizer=opts[opt], metrics=['mean_squared_error'])
hist = model.fit(X_train.values, y_train, batch_size=batch_size, epochs=n_epochs,
validation_data=(X_val.values, y_val), verbose=1,     callbacks=callbacks)
print(hist.history)     best_epoch = np.argmax(hist.history['accuracy'])
print(best_epoch)     best_acc = hist.history['accuracy'][best_epoch]
print(best_acc)     best_model = create_model(opt)     best_model.summary()
   # Load the model weights with the highest validation accuracy
best_model.load_weights('checkpoints/optimizers_best_' + opt + '.h5')
best_model.compile(loss='mse', optimizer=opts[opt],
metrics=['accuracy'])     score = best_model.evaluate(X_test.values,
y_test, verbose=0)     results.append([opt, best_epoch, best_acc,
score[1]]) res = pd.DataFrame(results) res


res.columns = ['optimizer', 'epochs', 'val_accuracy',
'test_accuracy'] res
```

Output:

	0	1	2	3
0	sgd	0	0.0	0.0
1	sgd-0001	0	0.0	0.0
2	adam	0	0.0	0.0
3	adadelta	0	0.0	0.0
4	rmsprop	0	0.0	0.0
5	rmsprop-0001	0	0.0	0.0
6	nadam	0	0.0	0.0
7	adamax	0	0.0	0.0

	optimizer	epochs	val_accuracy	test_accuracy
0	rmsprop	216	0.574219	0.571875
1	adamax	251	0.585938	0.603125
2	sgd-0001	167	0.562500	0.571875
3	nadam	133	0.582031	0.553125
4	adam	139	0.578125	0.581250
5	sgd	0	0.000000	0.000000
6	rmsprop-0001	62	0.550781	0.565625
7	adadelta	208	0.578125	0.575000

Result:

Ex-9 Improving Generalization with Regularization

Aim: Implement a python program for Improving generalizations with regularization.

Algorithm:

1. Define a neural network architecture with a set of parameters that need to be learned through training.

2. Split the dataset into training and validation sets.

3. Choose a suitable regularization technique, such as L1, L2, or Dropout regularization
4. Initialize the weights and biases of the network randomly.
5. Set the number of epochs and the learning rate for training the model.
6. For each epoch, perform the following s:
    a. Feed the training data through the network and compute the loss using a suitable loss function.
    b. Add the regularization term to the loss function.
    c. Use backpropagation to calculate the gradients of the loss with respect to the weights.
    d. Update the weights using the gradients and the learning rate.
    e. Evaluate the performance of the model on the validation set.
7. If the performance on the validation set does not improve for a certain number of epochs, stop the training and return the current model.
8. Otherwise, continue training until the desired level of performance is achieved.
9. Once the training is complete, use the trained model to make predictions on new data

```
import numpy as np import pandas as
pd from matplotlib import pyplot as
plt

from keras.models import Sequential
from keras.layers import Dense,
Dropout from keras import regularizers


# Dataset can be downloaded at https://archive.ics.uci.edu/ml/machine-learning-databases/00275/
data = pd.read_csv('/content/hour.csv

                                                                                        ') data
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
0	1	2011-01-01	1	0	1	0	0	6	0	
1	2	2011-01-01	1	0	1	1	0	6	0	
2	3	2011-01-01	1	0	1	2	0	6	0	
3	4	2011-01-01	1	0	1	3	0	6	0	
4	5	2011-01-01	1	0	1	4	0	6	0	
...	...	...	...	...	...	...	...	...	...	.
17374	17375	2012-12-31	1	1	12	19	0	1	1	2
17375	17376	2012-12-31	1	1	12	20	0	1	1	2

```
# Feature engineering ohe_features = ['season', 'weathersit',
'mnth', 'hr', 'weekday'] for feature in ohe_features:
    dummies = pd.get_dummies(data[feature], prefix=feature, drop_first=False)

data = pd.concat([data, dummies], axis=1) data
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	...	hr_21	hr_22	hr_23	weekday_0
0	1	2011-01-01	1 0	0 0	1	0	0	6	0	1	...	0	0		
1	2	2011-01-01	1 0	0 0	1	1	0	6	0	1	...	0	0		
2	3	2011-01-01	1 0	0 0	1	2	0	6	0	1	...	0	0		
3	4	2011-01-01	1 0	0 0	1	3	0	6	0	1	...	0	0		
4	5	2011-01-01	1 0	0 0	1	4	0	6	0	1	...	0	0		
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	17375 0	2012-12-31	1 0	1	12	19	0	1	1	2	...	0	0		
17375	17376 0	2012-12-31	1 0	1	12	20	0	1	1	2	...	0	0		

2012-

17376	17377 0	1 0 12-31	1	12	21	0	1	1	1	...	1	0
17377	17378 0	2012- 1 0 12-31	1	12	22	0	1	1	1	...	0	1
17378	17379 1	2012- 1 0 12-31	1	12	23	0	1	1	1	...	0	0

17379 rows × 68 columns

```
drop_features = ['instant', 'dteday', 'season', 'weathersit',
                 'weekday', 'atemp', 'mnth', 'workingday', 'hr', 'casual', 'registered']
data = data.drop(drop_features, axis=1)
data
```

	yr	holiday	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	...	hr_21	hr_22	hr_23	weekday_0	w
0	0	0 0	0.24 0	0.81	0.0000	16	1	0	0	0	...	0	0			
1	0	0 0	0.22 0	0.80	0.0000	40	1	0	0	0	...	0	0			
2	0	0 0	0.22 0	0.80	0.0000	32	1	0	0	0	...	0	0			
3	0	0 0	0.24 0	0.75	0.0000	13	1	0	0	0	...	0	0			
4	0	0 0	0.24 0	0.75	0.0000	1	1	0	0	0	...	0	0			
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	1	0 0	0.26 0	0.60	0.1642	119	1	0	0	0	...	0	0			
17375	1	0 0	0.26 0	0.60	0.1642	89	1	0	0	0	...	0	0			
17376	1	0 0	0.26 0	0.60	0.1642	90	1	0	0	0	...	1	0			
17377	1	0 0	0.26 0	0.56	0.1343	61	1	0	0	0	...	0	1			
17378	1	0 1	0.26 0	0.65	0.1343	49	1	0	0	0	...	0	0			

17379 rows × 57 columns

```
norm_features = ['cnt', 'temp', 'hum', 'windspeed']
scaled_features = {} for feature in norm_features:
    mean, std = data[feature].mean(), data[feature].std()
scaled_features[feature] = [mean, std]     data.loc[:,
feature] = (data[feature] - mean)/std data
```

	yr	holiday	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	...	hr_21	hr_22	hr_2
0	0	0	-1.334609	0.947345	-1.553844	-0.956312	1	0	0	0	...	0	0	
1	0	0	-1.438475	0.895513	-1.553844	-0.823998	1	0	0	0	...	0	0	
2	0	0	-1.438475	0.895513	-1.553844	-0.868103	1	0	0	0	...	0	0	
3	0	0	-1.334609	0.636351	-1.553844	-0.972851	1	0	0	0	...	0	0	
4	0	0	-1.334609	0.636351	-1.553844	-1.039008	1	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
17374	1	0	-1.230743	-0.141133	-0.211685	-0.388467	1	0	0	0	...	0	0	
17375	1	0	-1.230743	-0.141133	-0.211685	-0.553859	1	0	0	0	...	0	0	
17376	1	0	-1.230743	-0.141133	-0.211685	-0.548346	1	0	0	0	...	1	0	

17377	1	0	-1.230743	-0.348463	-0.456086	-0.708224	1	0	0	...	0	1
17378	1	0	-1.230743	0.118028	-0.456086	-0.774381	1	0	0	...	0	0

17379 rows × 57 columns

```
data[-31*24:]
```

	yr	holiday	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	...	hr_21	hr_22	hr_2
16635	1	0	-1.023012	0.636351	-1.553844	-0.145892	0	0	0	1	...	0	1	
16636	1	0	-1.023012	0.636351	-0.821460	-0.438084	0	0	0	1	...	0	0	
16637	1	0	-1.230743	0.947345	-1.553844	-0.449111	0	0	0	1	...	0	0	
16638	1	0	-1.230743	0.947345	-1.553844	-0.664120	0	0	0	1	...	0	0	
16639	1	0	-1.230743	0.947345	-1.553844	-0.768868	0	0	0	1	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
17374	1	0	-1.230743	-0.141133	-0.211685	-0.388467	1	0	0	0	...	0	0	
17375	1	0	-1.230743	-0.141133	-0.211685	-0.553859	1	0	0	0	...	0	0	
17376	1	0	-1.230743	-0.141133	-0.211685	-0.548346	1	0	0	0	...	1	0	
17377	1	0	-1.230743	-0.348463	-0.456086	-0.708224	1	0	0	0	...	0	1	
17378	1	0	-1.230743	0.118028	-0.456086	-0.774381	1	0	0	0	...	0	0	

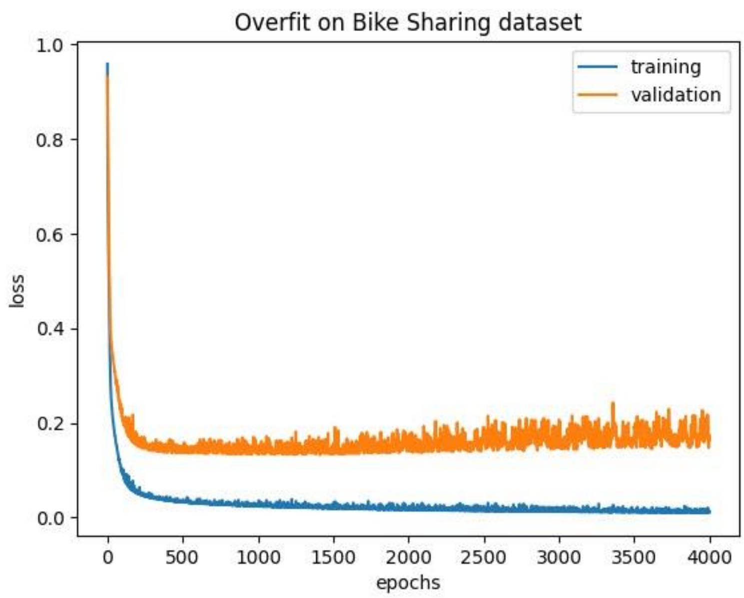
744 rows × 57 columns

```
# Save the final month for testing
# 744 rows
test_data = data[-31*24:]
data = data[:-31*24]
# Extract the target field
target_fields = ['cnt']
features, targets = data.drop(target_fields, axis=1), data[target_fields]
test_features, test_targets = test_data.drop(target_fields, axis=1), test_data[target_fields]
# Create a validation set (based on the last )
X_train, y_train = features[:-30*24], targets[:-30*24]
X_val, y_val = features[-30*24:], targets[-30*24:]
```

```
model = Sequential()
model.add(Dense(250, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(1, activation='linear'))
# Compile model
model.compile(loss='mse', optimizer='sgd', metrics=['mse'])
```

```
n_epochs = 4000
batch_size = 1024

history = model.fit(X_train.values, y_train['cnt'],
validation_data=(X_val.values, y_val['cnt']),
batch_size=batch_size, epochs=n_epochs, verbose=0
                )
```

```
plt.plot(np.arange(len(history.history['loss'])), history.history['loss'], label='training')
plt.plot(np.arange(len(history.history['val_loss'])), history.history['val_loss'],
label='validation') plt.title('Overfit on Bike Sharing dataset') plt.xlabel('epochs')
plt.ylabel('loss') plt.legend(loc=0) plt.show()
```



```
print('Minimum loss: ', min(history.history['val_loss']),
      '\nAfter ', np.argmin(history.history['val_loss']), ' epochs')

# Minimum loss:  0.140975862741
# After  730  epochs
```

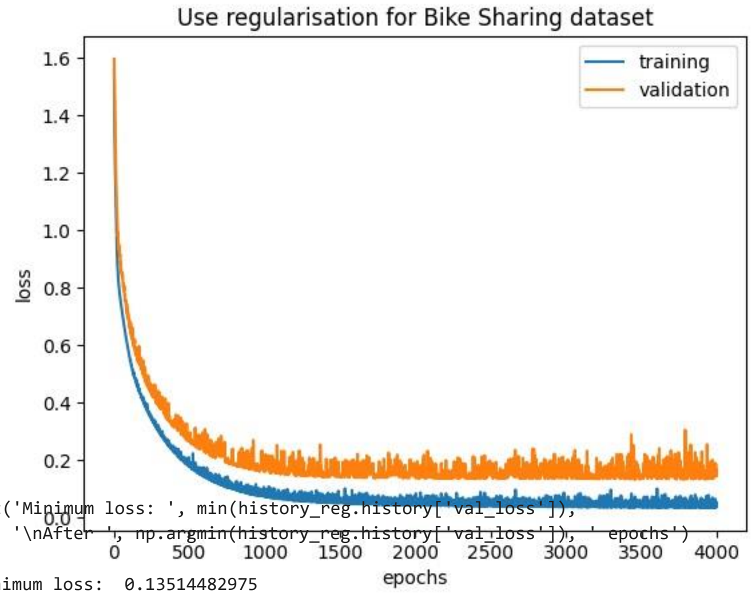
```
Minimum loss:  0.13224484026432037
After  1245  epochs
```

```
model_reg = Sequential()
model_reg.add(Dense(250, input_dim=X_train.shape[1], activation='relu',
kernel_regularizer=regularizers.l2(0.005))) model_reg.add(Dense(150, activation='relu')) model_reg.add(Dense(50,
activation='relu')) model_reg.add(Dense(25, activation='relu',
kernel_regularizer=regularizers.l2(0.005))) model_reg.add(Dense(1, activation='linear'))

# Compile model model_reg.compile(loss='mse', optimizer='sgd',
metrics=['mse'])


history_reg = model_reg.fit(X_train.values, y_train['cnt'],
validation_data=(X_val.values, y_val['cnt']),
batch_size=batch_size, epochs=n_epochs, verbose=1
                )
```

Streaming output truncated to the last 5000 lines.

```
Epoch 1501/4000
16/16 [==============================] - 0s 15ms/step - loss: 0.0692 - mse: 0.0464 - val_loss: 0.1475 - val_mse: 0.1247
Epoch 1502/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0659 - mse: 0.0431 - val_loss: 0.1418 - val_mse: 0.1190
Epoch 1503/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0521 - mse: 0.0293 - val_loss: 0.1446 - val_mse: 0.1218
Epoch 1504/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0530 - mse: 0.0302 - val_loss: 0.1463 - val_mse: 0.1235
Epoch 1505/4000
16/16 [==============================] - 0s 20ms/step - loss: 0.0619 - mse: 0.0392 - val_loss: 0.1451 - val_mse: 0.1223
Epoch 1506/4000
16/16 [==============================] - 0s 26ms/step - loss: 0.0689 - mse: 0.0462 - val_loss: 0.1469 - val_mse: 0.1242
Epoch 1507/4000
16/16 [==============================] - 0s 25ms/step - loss: 0.0588 - mse: 0.0361 - val_loss: 0.1435 - val_mse: 0.1207
Epoch 1508/4000
16/16 [==============================] - 0s 22ms/step - loss: 0.0555 - mse: 0.0328 - val_loss: 0.1442 - val_mse: 0.1215
Epoch 1509/4000
16/16 [==============================] - 0s 23ms/step - loss: 0.0542 - mse: 0.0315 - val_loss: 0.1417 - val_mse: 0.1190
Epoch 1510/4000
16/16 [==============================] - 0s 26ms/step - loss: 0.0602 - mse: 0.0375 - val_loss: 0.1454 - val_mse: 0.1228
Epoch 1511/4000
16/16 [==============================] - 0s 22ms/step - loss: 0.0615 - mse: 0.0389 - val_loss: 0.1421 - val_mse: 0.1195
Epoch 1512/4000
16/16 [==============================] - 0s 21ms/step - loss: 0.0590 - mse: 0.0364 - val_loss: 0.1456 - val_mse: 0.1230
Epoch 1513/4000
16/16 [==============================] - 0s 17ms/step - loss: 0.0571 - mse: 0.0345 - val_loss: 0.1431 - val_mse: 0.1205
Epoch 1514/4000
16/16 [==============================] - 0s 18ms/step - loss: 0.0676 - mse: 0.0450 - val_loss: 0.1452 - val_mse: 0.1226
Epoch 1515/4000
16/16 [==============================] - 0s 15ms/step - loss: 0.0629 - mse: 0.0403 - val_loss: 0.1492 - val_mse: 0.1266
Epoch 1516/4000
16/16 [==============================] - 0s 18ms/step - loss: 0.0571 - mse: 0.0345 - val_loss: 0.1425 - val_mse: 0.1200
Epoch 1517/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0508 - mse: 0.0283 - val_loss: 0.1539 - val_mse: 0.1313
Epoch 1518/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0523 - mse: 0.0298 - val_loss: 0.1729 - val_mse: 0.1504
Epoch 1519/4000
16/16 [==============================] - 0s 17ms/step - loss: 0.0597 - mse: 0.0372 - val_loss: 0.1945 - val_mse: 0.1720
Epoch 1520/4000
16/16 [==============================] - 0s 17ms/step - loss: 0.0685 - mse: 0.0461 - val_loss: 0.1694 - val_mse: 0.1469
Epoch 1521/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0586 - mse: 0.0361 - val_loss: 0.1851 - val_mse: 0.1626
Epoch 1522/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0594 - mse: 0.0369 - val_loss: 0.1799 - val_mse: 0.1575
Epoch 1523/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0560 - mse: 0.0335 - val_loss: 0.1667 - val_mse: 0.1443
Epoch 1524/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0544 - mse: 0.0320 - val_loss: 0.1571 - val_mse: 0.1347
Epoch 1525/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0511 - mse: 0.0287 - val_loss: 0.1736 - val_mse: 0.1512
Epoch 1526/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0690 - mse: 0.0466 - val_loss: 0.1704 - val_mse: 0.1481
Epoch 1527/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0645 - mse: 0.0422 - val_loss: 0.1898 - val_mse: 0.1674
Epoch 1528/4000
16/16 [==============================] - 0s 16ms/step - loss: 0.0579 - mse: 0.0356 - val_loss: 0.1695 - val_mse: 0.1472
Epoch 1529/4000
```

```
plt.plot(np.arange(len(history_reg.history['loss'])), history_reg.history['loss'], label='training')
plt.plot(np.arange(len(history_reg.history['val_loss'])), history_reg.history['val_loss'],
label='validation') plt.title('Use regularisation for Bike Sharing dataset') plt.xlabel('epochs')
plt.ylabel('loss') plt.legend(loc=0) plt.show()
```




```
print('Minimum loss: ', min(history_reg.history['val_loss']),
      '\nAfter ', np.argmin(history_reg.history['val_loss']), ' epochs')

# Minimum loss:  0.13514482975
# After  3647  epochs
```

```
Minimum loss:  0.13387779891490936
After  3458  epochs
```

Ex-10 Adding dropout to prevent overfitting

Aim: To implement improving generalisation with regularisation.

Algorithm:

1. Initialize your neural network structure.
2. Choose a dropout rate (e.g., 0.2 to 0.5), which represents the fraction of neurons to "turn off" during training.
3. During the training process:
4. For each layer where dropout is applied:
5. Randomly set a fraction of the neurons to zero (dropout) by creating a binary mask.
6. Multiply the input to that layer by this mask to deactivate some neurons.
7. Continue with forward and backward propagation as usual, taking into account the dropped neurons.
8. During evaluation (not training), don't use dropout. Instead, scale the neuron activations by (1 - dropout_rate) to maintain expected values.
9. Repeat the training process for multiple epochs while adjusting other training parameters as needed.

Adding dropout to prevent over tting

Another popular method for regularization is dropout.

A dropout forces a neural network to learn multiple independent representations by randomly removing connections between neurons in the learning phase.

For example, when using a dropout of 0.5, the network has to see each example twice before the connection is learned.

Therefore, a network with dropout can be seen as an ensemble of networks.

Using the below code we will improve a model that clearly over ts the training data by adding dropouts.

```
import numpy as np import pandas as
pd from matplotlib import pyplot as
plt

from keras.models import Sequential
from keras.layers import Dense,
Dropout

import numpy as np
from matplotlib import pyplot as plt


# Dataset can be downloaded at https://archive.ics.uci.edu/ml/machine-learning-databases/00275/
```

```
data = pd.read_csv('/content/hour.csv')
data
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0000	3
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0000	8
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0000	5
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0000	3
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0000	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	17375	2012-12-31	1	1	12	19	0	1	1	2	0.26	0.2576	0.60	0.1642	11
17375	17376	2012-12-31	1	1	12	20	0	1	1	2	0.26	0.2576	0.60	0.1642	8

2012

```
# Feature engineering ohe_features = ['season', 'weathersit',
'mnth', 'hr', 'weekday'] for feature in ohe_features:
    dummies = pd.get_dummies(data[feature], prefix=feature, drop_first=False)

data = pd.concat([data, dummies], axis=1) data
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	...	hr_21	hr_22	hr_23	weekday_0
0	1	2011-01-01	1	0	1	0	0	6	0	1	...	0	0	0	0
		2011-													

1	2	2011-01-01	1	0	1	1	0	6	0	1	...	0	0	0
2	3	2011-01-01	1	0	1	2	0	6	0	1	...	0	0	0
3	4	2011-01-01	1	0	1	3	0	6	0	1	...	0	0	0
4	5	2011-01-01	1	0	1	4	0	6	0	1	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	17375	2012-12-31	1	1	12	19	0	1	1	2	...	0	0	0
17375	17376	2012-12-31	1	1	12	20	0	1	1	2	...	0	0	0
17376	17377	2012-12-31	1	1	12	21	0	1	1	1	...	0	0	0
17377	17378	2012-12-31	1	1	12	22	0	1	1	1	...	0	1	0
17378	17379	2012-12-31	1	1	12	23	0	1	1	1	...	0	0	1

17379 rows × 68 columns

```
drop_features = ['instant', 'dteday', 'season', 'weathersit', 'weekday', 'atemp', 'mnth', 'workingday', 'hr', 'casual', 'registered'] data = data.drop(drop_features, axis=1)
data
```

	yr	**holiday**	**temp**	**hum**	**windspeed**	**cnt**	**season_1**	**season_2**	**season_3**	**season_4**	**...**	**hr_21**	**hr_22**	**hr_23**	**weekday_0**
0	0	0	0.24	0.81	0.0000	16	1	0	0	0	...	0	0	0	0
1	0	0	0.22	0.80	0.0000	40	1	0	0	0	...	0	0	0	0
2	0	0	0.22	0.80	0.0000	32	1	0	0	0	...	0	0	0	0
3	0	0	0.24	0.75	0.0000	13	1	0	0	0	...	0	0	0	0
4	0	0	0.24	0.75	0.0000	1	1	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	1	0	0.26	0.60	0.1642	119	1	0	0	0	...	0	0	0	0
17375	1	0	0.26	0.60	0.1642	89	1	0	0	0	...	0	0	0	0
17376	1	0	0.26	0.60	0.1642	90	1	0	0	0	...	1	0	0	0
17377	1	0	0.26	0.56	0.1343	61	1	0	0	0	...	0	1	0	0
17378	1	0	0.26	0.65	0.1343	49	1	0	0	0	...	0	0	1	0

17379 rows × 57 columns

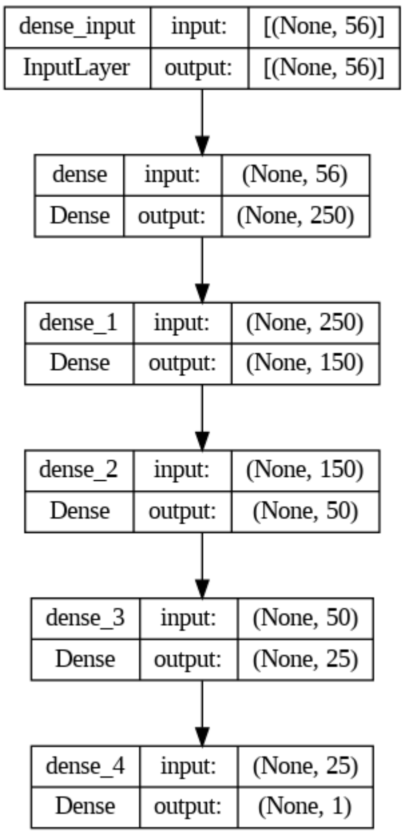
```
norm_features = ['cnt', 'temp', 'hum', 'windspeed']
scaled_features = {} for feature in norm_features:
    mean, std = data[feature].mean(), data[feature].std()
scaled_features[feature] = [mean, std]     data.loc[:,
feature] = (data[feature] - mean)/std scaled_features
```

```
{'cnt': [189.46308763450142, 181.38759909186473],
 'temp': [0.4969871684216583, 0.19255612124972219],
 'hum': [0.6272288394038783, 0.19292983406291508],
 'windspeed': [0.1900976063064618, 0.12234022857279049]}
```

```
# Save the final month for testing
test_data = data[-31*24:] data =
data[:-31*24]

# Extract the target field target_fields = ['cnt'] features, targets =
data.drop(target_fields, axis=1), data[target_fields] test_features, test_targets =
test_data.drop(target_fields, axis=1), test_data[target_fields]

# Create a validation set (based on the last )
X_train, y_train = features[:-30*24], targets[:-30*24]
X_val, y_val = features[-30*24:], targets[-30*24:]
```

```
model = Sequential() model.add(Dense(250,
input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(150, activation='relu')) model.add(Dense(50,
```

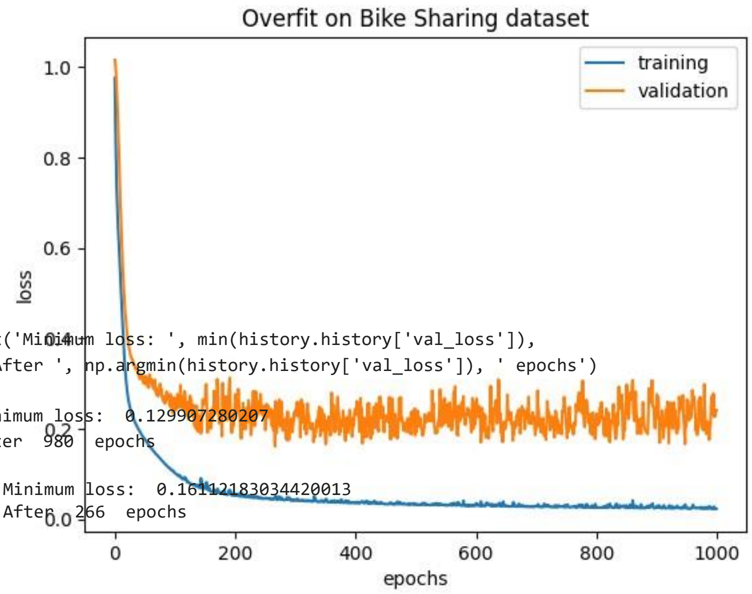
```
activation='relu')) model.add(Dense(25, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile model model.compile(loss='mse', optimizer='sgd',
metrics=['mse'])


model.summary()
     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
     dense (Dense)               (None, 250)               14250
     dense_1 (Dense)             (None, 150)               37650
     dense_2 (Dense)             (None, 50)                7550
     dense_3 (Dense)             (None, 25)                1275
     dense_4 (Dense)             (None, 1)                 26
     =================================================================
     Total params: 60751 (237.31 KB)
     Trainable params: 60751 (237.31 KB)
     Non-trainable params: 0 (0.00 Byte)
     _________________________________________________________________


!pip install pydot

     Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages (1.4.2)
     Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.10/dist-packages (from pydot) (3.1.1)


# Visualize network architecture

import pydot import pydotplus
import graphviz from
IPython.display import SVG
#from tensorflow.keras.utils.vis_utils import model_to_dot
#from tensorflow.keras.utils.vis_utils import plot_model
from tensorflow.keras.utils import model_to_dot from
tensorflow.keras.utils import plot_model

SVG(model_to_dot(model, show_shapes=True).create(prog="dot", format="svg"))

# Save the visualization as a file plot_model(model, show_shapes=True,
to_file="dropout_network_model.png")
```




```
n_epochs = 1000
batch_size = 1024

history = model.fit(X_train.values, y_train['cnt'],
validation_data=(X_val.values, y_val['cnt']),
batch_size=batch_size, epochs=n_epochs, verbose=1
    )
```

```
Epoch 1/1000
16/16 [==============================] - 1s 36ms/step - loss: 0.9760 - mse: 0.9760 - val_loss: 1.0160 - val_mse: 1.0160
Epoch 2/1000
16/16 [==============================] - 0s 18ms/step - loss: 0.8708 - mse: 0.8708 - val_loss: 1.0062 - val_mse: 1.0062
Epoch 3/1000
16/16 [==============================] - 0s 15ms/step - loss: 0.7940 - mse: 0.7940 - val_loss: 0.9866 - val_mse: 0.9866
Epoch 4/1000
16/16 [==============================] - 0s 15ms/step - loss: 0.7348 - mse: 0.7348 - val_loss: 0.9558 - val_mse: 0.9558
Epoch 5/1000
16/16 [==============================] - 0s 14ms/step - loss: 0.6912 - mse: 0.6912 - val_loss: 0.9267 - val_mse: 0.9267
Epoch 6/1000
16/16 [==============================] - 0s 13ms/step - loss: 0.6571 - mse: 0.6571 - val_loss: 0.8931 - val_mse: 0.8931
Epoch 7/1000
16/16 [==============================] - 0s 14ms/step - loss: 0.6280 - mse: 0.6280 - val_loss: 0.8444 - val_mse: 0.8444
Epoch 8/1000
16/16 [==============================] - 0s 13ms/step - loss: 0.5996 - mse: 0.5996 - val_loss: 0.8122 - val_mse: 0.8122
Epoch 9/1000
16/16 [==============================] - 0s 16ms/step - loss: 0.5714 - mse: 0.5714 - val_loss: 0.7582 - val_mse: 0.7582
Epoch 10/1000
16/16 [==============================] - 0s 20ms/step - loss: 0.5430 - mse: 0.5430 - val_loss: 0.7042 - val_mse: 0.7042
Epoch 11/1000
16/16 [==============================] - 0s 14ms/step - loss: 0.5141 - mse: 0.5141 - val_loss: 0.6735 - val_mse: 0.6735
Epoch 12/1000
16/16 [==============================] - 0s 13ms/step - loss: 0.4855 - mse: 0.4855 - val_loss: 0.6383 - val_mse: 0.6383
Epoch 13/1000
16/16 [==============================] - 0s 14ms/step - loss: 0.4567 - mse: 0.4567 - val_loss: 0.5970 - val_mse: 0.5970
Epoch 14/1000
16/16 [==============================] - 0s 23ms/step - loss: 0.4291 - mse: 0.4291 - val_loss: 0.5637 - val_mse: 0.5637
Epoch 15/1000
16/16 [==============================] - 0s 23ms/step - loss: 0.4032 - mse: 0.4032 - val_loss: 0.5342 - val_mse: 0.5342
Epoch 16/1000
16/16 [==============================] - 0s 22ms/step - loss: 0.3784 - mse: 0.3784 - val_loss: 0.5077 - val_mse: 0.5077
Epoch 17/1000
16/16 [==============================] - 0s 23ms/step - loss: 0.3554 - mse: 0.3554 - val_loss: 0.4838 - val_mse: 0.4838
Epoch 18/1000
16/16 [==============================] - 0s 20ms/step - loss: 0.3352 - mse: 0.3352 - val_loss: 0.4611 - val_mse: 0.4611
Epoch 19/1000
16/16 [==============================] - 0s 22ms/step - loss: 0.3175 - mse: 0.3175 - val_loss: 0.4473 - val_mse: 0.4473
Epoch 20/1000
16/16 [==============================] - 1s 38ms/step - loss: 0.3021 - mse: 0.3021 - val_loss: 0.4327 - val_mse: 0.4327
Epoch 21/1000
16/16 [==============================] - 0s 27ms/step - loss: 0.2890 - mse: 0.2890 - val_loss: 0.4124 - val_mse: 0.4124
Epoch 22/1000
16/16 [==============================] - 0s 28ms/step - loss: 0.2778 - mse: 0.2778 - val_loss: 0.4042 - val_mse: 0.4042
Epoch 23/1000
16/16 [==============================] - 0s 21ms/step - loss: 0.2687 - mse: 0.2687 - val_loss: 0.3952 - val_mse: 0.3952
Epoch 24/1000
16/16 [==============================] - 0s 25ms/step - loss: 0.2608 - mse: 0.2608 - val_loss: 0.3850 - val_mse: 0.3850
Epoch 25/1000
16/16 [==============================] - 0s 27ms/step - loss: 0.2543 - mse: 0.2543 - val_loss: 0.3755 - val_mse: 0.3755
Epoch 26/1000
16/16 [==============================] - 0s 23ms/step - loss: 0.2483 - mse: 0.2483 - val_loss: 0.3718 - val_mse: 0.3718
Epoch 27/1000
16/16 [==============================] - 0s 28ms/step - loss: 0.2426 - mse: 0.2426 - val_loss: 0.3629 - val_mse: 0.3629
Epoch 28/1000 16/16 [==============================] - 0s 27ms/step - loss: 0.2378 - mse: 0.2378 - val_loss: 0.3606 -
val_mse: 0.3606
Epoch 29/1000
16/16 [==============================] - 0s 26ms/step - loss: 0.2332 - mse: 0.2332 - val_loss: 0.3534 - val_mse: 0.3534
```

```
plt.plot(np.arange(len(history.history['loss'])), history.history['loss'], label='training')
plt.plot(np.arange(len(history.history['val_loss'])), history.history['val_loss'],
label='validation') plt.title('Overfit on Bike Sharing dataset') plt.xlabel('epochs')
plt.ylabel('loss') plt.legend(loc=0) plt.show()
```




```
print('Minimum loss: ', min(history.history['val_loss']),
 '\nAfter ', np.argmin(history.history['val_loss']), ' epochs')

# Minimum loss:  0.129907280207
# After  980  epochs
```

```
Minimum loss:  0.16112183034420013
After  266  epochs
```

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for speci c features, providing some specialization. Neighboring neurons come to rely on this specialization, which, if taken too far, can result in a fragile model too specialized for the training data. This reliance on context for a neuron during training is referred to as complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the speci c weights of neurons. This, in turn, results in a network capable of better generalization and less likely to over t the training data.

```
model_drop = Sequential() model_drop.add(Dense(250,
input_dim=X_train.shape[1], activation='relu'))
model_drop.add(Dropout(0.20))
model_drop.add(Dense(150, activation='relu'))
model_drop.add(Dropout(0.20))
model_drop.add(Dense(50, activation='relu'))
model_drop.add(Dropout(0.20))
model_drop.add(Dense(25, activation='relu'))
model_drop.add(Dropout(0.20))
model_drop.add(Dense(1, activation='linear'))

# Compile model model_drop.compile(loss='mse', optimizer='sgd',
metrics=['mse'])
```

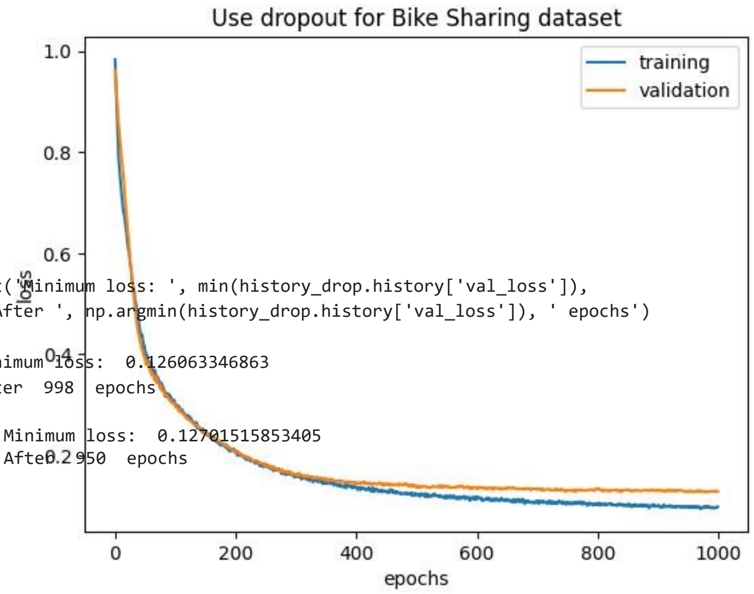
```
model_drop.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 250)               14250
 dropout (Dropout)           (None, 250)               0
 dense_6 (Dense)             (None, 150)               37650
 dropout_1 (Dropout)         (None, 150)               0
 dense_7 (Dense)             (None, 50)                7550
 dropout_2 (Dropout)         (None, 50)                0
 dense_8 (Dense)             (None, 25)                1275
 dropout_3 (Dropout)         (None, 25)                0
 dense_9 (Dense)             (None, 1)                 26
=================================================================
Total params: 60751 (237.31 KB)
Trainable params: 60751 (237.31 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history_drop = model_drop.fit(X_train.values, y_train['cnt'],
validation_data=(X_val.values, y_val['cnt']),
batch_size=batch_size, epochs=n_epochs, verbose=1
    )
```

```
Epoch 1/1000
16/16 [==============================] - 1s 30ms/step - loss: 0.9834 - mse: 0.9834 - val_loss: 0.9624 - val_mse: 0.9624
Epoch 2/1000
16/16 [==============================] - 0s 30ms/step - loss: 0.9488 - mse: 0.9488 - val_loss: 0.9457 - val_mse: 0.9457
Epoch 3/1000
16/16 [==============================] - 0s 22ms/step - loss: 0.9151 - mse: 0.9151 - val_loss: 0.9258 - val_mse: 0.9258
Epoch 4/1000
16/16 [==============================] - 0s 20ms/step - loss: 0.8757 - mse: 0.8757 - val_loss: 0.9064 - val_mse: 0.9064
Epoch 5/1000
16/16 [==============================] - 1s 32ms/step - loss: 0.8455 - mse: 0.8455 - val_loss: 0.8881 - val_mse: 0.8881
Epoch 6/1000
16/16 [==============================] - 1s 33ms/step - loss: 0.8177 - mse: 0.8177 - val_loss: 0.8701 - val_mse: 0.8701
Epoch 7/1000
16/16 [==============================] - 1s 45ms/step - loss: 0.7856 - mse: 0.7856 - val_loss: 0.8539 - val_mse: 0.8539
Epoch 8/1000
16/16 [==============================] - 0s 30ms/step - loss: 0.7699 - mse: 0.7699 - val_loss: 0.8434 - val_mse: 0.8434
Epoch 9/1000
16/16 [==============================] - 0s 30ms/step - loss: 0.7549 - mse: 0.7549 - val_loss: 0.8259 - val_mse: 0.8259
Epoch 10/1000
16/16 [==============================] - 0s 23ms/step - loss: 0.7416 - mse: 0.7416 - val_loss: 0.8142 - val_mse: 0.8142
Epoch 11/1000
16/16 [==============================] - 0s 21ms/step - loss: 0.7306 - mse: 0.7306 - val_loss: 0.8012 - val_mse: 0.8012
Epoch 12/1000
16/16 [==============================] - 0s 19ms/step - loss: 0.7116 - mse: 0.7116 - val_loss: 0.7832 - val_mse: 0.7832
Epoch 13/1000
16/16 [==============================] - 0s 18ms/step - loss: 0.7059 - mse: 0.7059 - val_loss: 0.7782 - val_mse: 0.7782
Epoch 14/1000
16/16 [==============================] - 0s 19ms/step - loss: 0.6906 - mse: 0.6906 - val_loss: 0.7611 - val_mse: 0.7611
Epoch 15/1000
16/16 [==============================] - 0s 18ms/step - loss: 0.6803 - mse: 0.6803 - val_loss: 0.7509 - val_mse: 0.7509
Epoch 16/1000
16/16 [==============================] - 1s 33ms/step - loss: 0.6777 - mse: 0.6777 - val_loss: 0.7359 - val_mse: 0.7359
Epoch 17/1000
16/16 [==============================] - 1s 43ms/step - loss: 0.6690 - mse: 0.6690 - val_loss: 0.7162 - val_mse: 0.7162
Epoch 18/1000
16/16 [==============================] - 1s 31ms/step - loss: 0.6598 - mse: 0.6598 - val_loss: 0.7032 - val_mse: 0.7032
Epoch 19/1000
16/16 [==============================] - 1s 35ms/step - loss: 0.6466 - mse: 0.6466 - val_loss: 0.6879 - val_mse: 0.6879
Epoch 20/1000
16/16 [==============================] - 1s 35ms/step - loss: 0.6429 - mse: 0.6429 - val_loss: 0.6730 - val_mse: 0.6730
Epoch 21/1000
16/16 [==============================] - 1s 33ms/step - loss: 0.6246 - mse: 0.6246 - val_loss: 0.6596 - val_mse: 0.6596
Epoch 22/1000
16/16 [==============================] - 1s 41ms/step - loss: 0.6174 - mse: 0.6174 - val_loss: 0.6467 - val_mse: 0.6467
Epoch 23/1000
16/16 [==============================] - 1s 33ms/step - loss: 0.6068 - mse: 0.6068 - val_loss: 0.6315 - val_mse: 0.6315
Epoch 24/1000
16/16 [==============================] - 1s 38ms/step - loss: 0.6041 - mse: 0.6041 - val_loss: 0.6208 - val_mse: 0.6208
Epoch 25/1000
16/16 [==============================] - 1s 37ms/step - loss: 0.5920 - mse: 0.5920 - val_loss: 0.6020 - val_mse: 0.6020
Epoch 26/1000 16/16 [==============================] - 1s 38ms/step - loss: 0.5802 - mse: 0.5802 - val_loss: 0.5899 -
val_mse: 0.5899
Epoch 27/1000
16/16 [==============================] - 1s 34ms/step - loss: 0.5728 - mse: 0.5728 - val_loss: 0.5747 - val_mse: 0.5747
Epoch 28/1000
16/16 [==============================] - 1s 37ms/step - loss: 0.5652 - mse: 0.5652 - val_loss: 0.5645 - val_mse: 0.5645
Epoch 29/1000
16/16 [==============================] - 1s 40ms/step - loss: 0.5465 - mse: 0.5465 - val_loss: 0.5503 - val_mse: 0.5503
```

```
plt.plot(np.arange(len(history_drop.history['loss'])), history_drop.history['loss'], label='training')
plt.plot(np.arange(len(history_drop.history['val_loss'])), history_drop.history['val_loss'],
label='validation') plt.title('Use dropout for Bike Sharing dataset') plt.xlabel('epochs') plt.ylabel('loss')
plt.legend(loc=0) plt.show()
```




```
print('Minimum loss: ', min(history_drop.history['val_loss']),
 '\nAfter ', np.argmin(history_drop.history['val_loss']), ' epochs')

# Minimum loss:  0.126063346863
# After  998  epochs
```

```
Minimum loss:  0.12701515853405
After  950  epochs
```

EX-11 IMAGE AGUMENTATIONT

Aim:To implement Image Agumentation in python language.

Algorithm:

1. Import the necessary libraries, including OpenCV (cv2), NumPy, and Matplotlib for visualization (optional).
2. Load the original image that you want to augment using OpenCV's cv2.imread().
3. To create augmented versions of the image, you can apply various transformations such as rotation, flipping, scaling, and brightness adjustments. Here are some common augmentations
4. Use Matplotlib or any other suitable library to display the augmented images for visual inspection and verification.
5. If you want to generate multiple augmented images, you can repeat s 3 and 4 within a loop, adjusting augmentation parameters as needed.
6. If you want to save the augmented images to disk for later use, use OpenCV's cv2.imwrite() function.
7. Use the augmented images along with the original images in your deep learning model's training dataset to increase diversity and improve model performance.
8. If you have multiple images to augment, repeat the above s for each image.
9. Experiment with different augmentation techniques and parameters to find the most effective augmentations for your specific problem.

```
pip install imgaug
```

```
Requirement already satisfied: imgaug in /usr/local/lib/python3.10/dist-packages (0.4.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.16.0)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.11.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from imgaug) (9.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from imgaug) (3.7.1)
Requirement already satisfied: scikit-image>=0.14.2 in /usr/local/lib/python3.10/dist-packages (from imgaug) (0.19.3)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (from imgaug) (4.8.0.76)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from imgaug) (2.31.3)
Requirement already satisfied: Shapely in /usr/local/lib/python3.10/dist-packages (from imgaug) (2.0.1)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (3.1)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (23.1
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (2.8.2)
```

```
pip install ipyplot
```

```
Requirement already satisfied: ipyplot in /usr/local/lib/python3.10/dist-packages (1.1.1)
Requirement already satisfied: IPython in /usr/local/lib/python3.10/dist-packages (from ipyplot) (7.34.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from ipyplot) (1.23.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from ipyplot) (9.4.0)
Requirement already satisfied: shortuuid in /usr/local/lib/python3.10/dist-packages (from ipyplot) (1.0.11)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (67.7.2)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (0.19.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from IPytho
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from IPython->ipyplot) (4.8.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->IPython->ipyplot) (
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->IPython->ipyplot) (0.7
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.
```

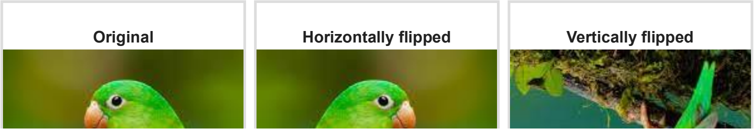
```
import imageio import imgaug as
ia import imgaug.augmenters as
iaa import ipyplot
```

```
input_img = imageio.imread('/content/b1.jpeg')
```

```
<ipython-input-23-d2130d9fdd9a>:1: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that o
input_img = imageio.imread('/content/b1.jpeg')
```

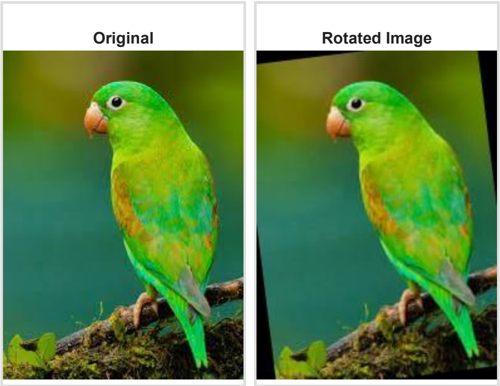
```
#Horizontal Flip hflip=
iaa.Fliplr(p=1.0) input_hf=
hflip.augment_image(input_img)
#Vertical Flip vflip= iaa.Flipud(p=1.0)
input_vf= vflip.augment_image(input_img) images_list=[input_img,
input_hf, input_vf] labels = ['Original', 'Horizontally flipped',
'Vertically flipped']
ipyplot.plot_images(images_list,labels=labels,img_width=180)
```

show html



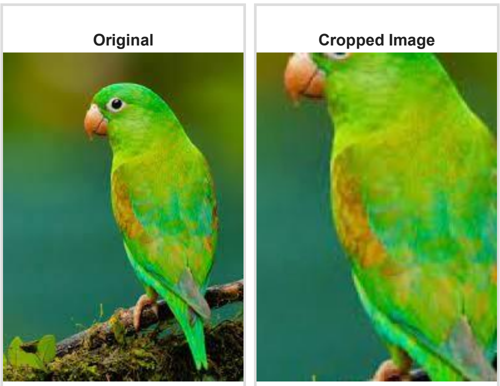

```
rot1 = iaa.Affine(rotate=(-90,20))
input_rot1 = rot1.augment_image(input_img)
images_list=[input_img, input_rot1]
labels = ['Original', 'Rotated Image']
ipyplot.plot_images(images_list,labels=labels,img_width=180)
```

show html



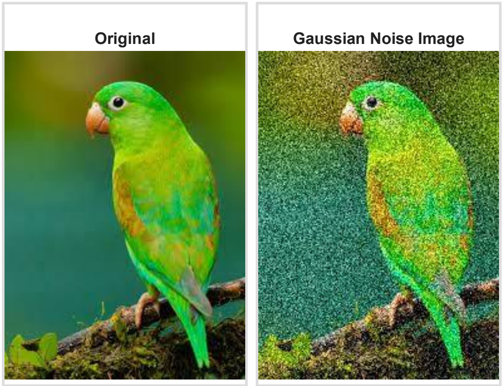

```
crop1 = iaa.Crop(percent=(0, 0.3))
input_crop1 = crop1.augment_image(input_img)
images_list=[input_img, input_crop1] labels = ['Original',
'Cropped Image']
ipyplot.plot_images(images_list,labels=labels,img_width=180
)
```

show html




```
noise=iaa.AdditiveGaussianNoise(10,40)
input_noise=noise.augment_image(input_img)
images_list=[input_img, input_noise] labels = ['Original',
'Gaussian Noise Image']
ipyplot.plot_images(images_list,labels=labels,img_width=180
)
```

show html




```
shear = iaa.Affine(shear=(-40,40))
input_shear=shear.augment_image(input_img
) images_list=[input_img, input_shear]
labels = ['Original', 'Image Shearing']
ipyplot.plot_images(images_list,labels=labels,img_width=180)
```

show html

Image Contrast This augmenter adjusts the image contrast by scaling pixel values.

```
contrast=iaa.GammaContrast((0.5, 2.0)) contrast_sig =
iaa.SigmoidContrast(gain=(5, 10), cutoff=(0.4, 0.6)) contrast_lin =
iaa.LinearContrast((0.6, 0.4)) input_contrast =
contrast.augment_image(input_img) sigmoid_contrast =
contrast_sig.augment_image(input_img) linear_contrast =
contrast_lin.augment_image(input_img) images_list=[input_img,
input_contrast,sigmoid_contrast,linear_contrast] labels = ['Original',
'Gamma Contrast','SigmoidContrast','LinearContrast']
ipyplot.plot_images(images_list,labels=labels,img_width=180)
```

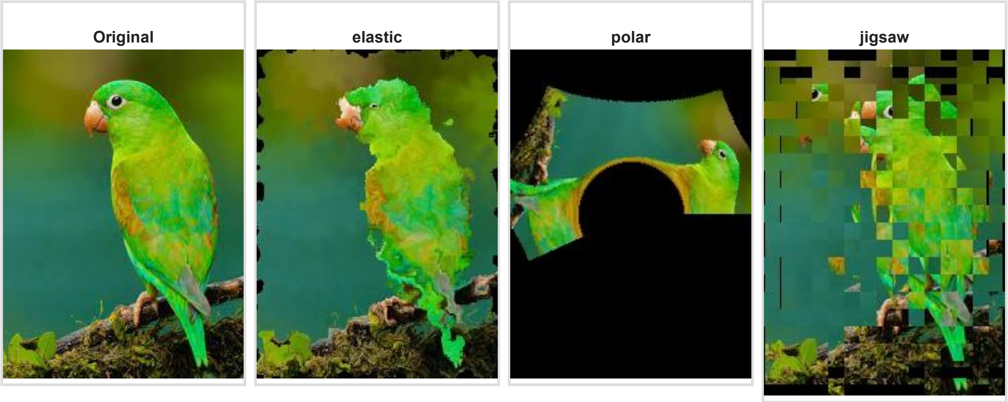
show html

Image Transformations The ‘Elastic Transformation’ augmenter transforms images by shifting pixels around locally using displacement elds. The augmenter’s parameters are alpha and sigma. The strength of the displacement is controlled by alpha, wherein greater values indicate that pixels are shifted further. The smoothness of the displacement is controlled by sigma, in which larger values result in smoother patterns.

```
elastic = iaa.ElasticTransformation(alpha=60.0, sigma=4.0) polar
= iaa.WithPolarWarping(iaa.CropAndPad(percent=(-0.2, 0.7)))
```

```
jigsaw = iaa.Jigsaw(nb_rows=20, nb_cols=15, max_steps=(3, 7))
input_elastic = elastic.augment_image(input_img) input_polar =
polar.augment_image(input_img) input_jigsaw =
jigsaw.augment_image(input_img) images_list=[input_img,
input_elastic,input_polar,input_jigsaw] labels = ['Original',
'elastic','polar','jigsaw']
ipyplot.plot_images(images_list,labels=labels,img_width=180)
```

show html



EX-12 Imagenet- AlexNet

Aim: To implement Imagenet- AlexNet in python language.

Algorithm

1. Import the necessary deep learning libraries (e.g., TensorFlow or PyTorch) and other supporting libraries.
2. Download and preprocess the ImageNet dataset or a subset of it.Normalize the images.Split the dataset into training, validation, and test sets.
3. Create a neural network model with the following layers:Convolutional layers with appropriate filter sizes, strides, and padding.Max-pooling layers, Fully connected (dense) layers, dropout layers to prevent overfitting. Define appropriate activation functions (e.g., ReLU) and batch normalization as needed.
4. Specify the loss function (e.g., categorical cross-entropy) and optimizer (e.g., SGD or Adam).Choose evaluation metrics (e.g., accuracy).
5. Apply data augmentation techniques such as random cropping, flipping, and rotation to increase the diversity of training examples.
6. Train the model on the training dataset using the compiled model, specifying the number of epochs, batch size, and other training parameters.Monitor training and validation performance to detect overfitting.
7. Fine-tune the model by adjusting hyperparameters or using learning rate schedules if necessary.
8. Evaluate the trained model on the test dataset to measure its performance in terms of accuracy or other relevant metrics.
9. Use the trained model to make predictions on new, unseen images.
10. Visualize model predictions and performance metrics.

```
import json
from torchvision.datasets.utils  import download_url


import torch
from torchvision  import models
from torchvision  import transforms
from PIL  import  Image

download_url ("https://s3.amazonaws.com/deep-learning-models/imag e-models/imagenet class index.json ", ".", "im


with open("imagenet_class_index.json" , "r") as h:
    labels = json.load (h)

alexnet = models.alexnet (pretrained=True)
```

```
Downloading https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json to ./i
100%|██████████| 35363/35363 [00:00<00:00, 2726729.40it/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'p
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/c
100%|██████████| 233M/233M [00:01<00:00, 143MB/s]
```

```
preprocess_image = transforms.Compose([
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(   mean=[0.485,
0.456, 0.406],   std=[0.229, 0.224,
0.225]
)])


image = Image.open("/content/cat.jpeg") image_tensor
= preprocess_image(image)



print(image_tensor.shape)

input_tensor = torch.unsqueeze(image_tensor, 0) print(input_tensor.shape)
```

```
torch.Size([3, 224, 224]) torch.Size([1,
3, 224, 224])
```

```
batch_t = torch.unsqueeze(image_tensor, 0) alexnet.eval()

with torch.no_grad():     predictedout = alexnet(batch_t)

label_indices = predictedout.argmax(dim=1) for p in label_indices:     print(labels[str(p.item())])
```

```
['n02124075' 'Egyptian cat']
```



```
[ n02124075 , Egyptian_cat ]
```


0s completed at 4:28 PM

EX-13 LSTM

Aim: To implement LSTM in python language.

Algorithm:

1. Import the deep learning framework you plan to use (e.g., TensorFlow or PyTorch) and other necessary libraries.
2. Load and preprocess your sequential data. LSTMs are commonly used for tasks like sequence prediction, text generation, or sentiment analysis.Preprocess the data, which may include tokenization, one-hot encoding, or embedding, depending on your task.
3. Create an LSTM model by defining the layers and their configurations.
    a. Specify the number of LSTM units (neurons) in each LSTM layer.
    b. Choose an appropriate activation function (usually 'tanh') for the LSTM cells.
    c. Optionally, stack multiple LSTM layers if needed.
    d. You can also add dropout layers to prevent overfitting.
4. Compile the LSTM model by specifying the loss function and optimizer.

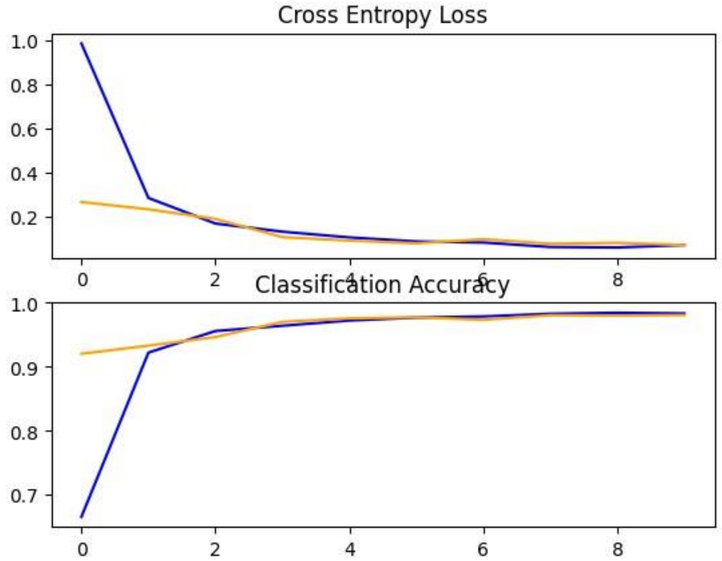
Choose appropriate metrics for evaluation (e.g., accuracy or mean squared error).

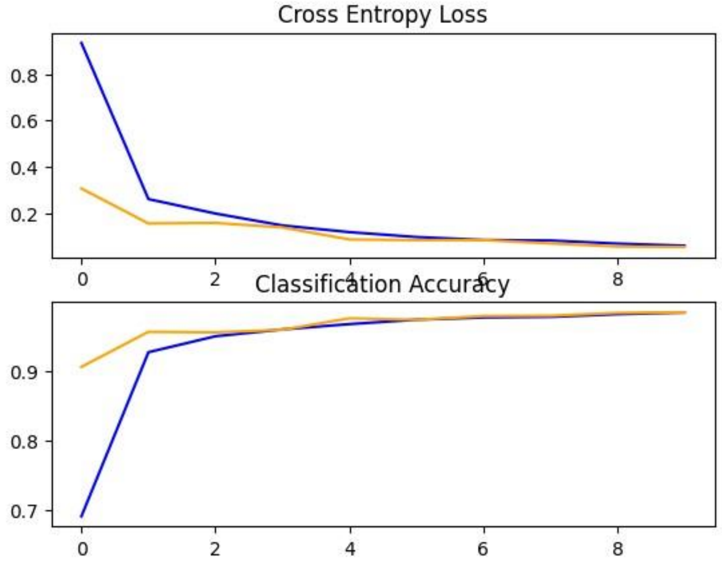
5. Train the LSTM model using your preprocessed data. Specify the number of epochs and batch size. Monitor training progress and adjust hyperparameters as needed.
6. After training, evaluate the LSTM model's performance on a validation or test dataset using relevant metrics.
7. Use the trained LSTM model to make predictions on new sequences or data points.
8. Visualize the model's predictions and performance metrics.

```
import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Dense, Dropout, LSTM from numpy import mean from numpy
import std from matplotlib import pyplot as plt from
sklearn.model_selection import KFold from
tensorflow.keras.datasets import mnist from
tensorflow.keras.utils import to_categorical


def load_dataset():
 # load dataset
 (trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel  trainX =
trainX.reshape((trainX.shape[0], 28, 28, 1))  testX =
testX.reshape((testX.shape[0], 28, 28, 1))
 # one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
return trainX, trainY, testX, testY


# scale pixels def prep_pixels(train,
test):  # convert from integers to
floats  train_norm =
train.astype('float32')  test_norm =
test.astype('float32')  # normalize
to range 0-1  train_norm =
train_norm / 255.0  test_norm =
test_norm / 255.0  # return
normalized images  return
train_norm, test_norm


def build_model():   model = Sequential()   model.add(LSTM(128, input_shape=((28,28)),
activation='relu', return_sequences=True))   model.add(Dropout(0.2))

  model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.1))

  model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

  model.add(Dense(10, activation='softmax'))   print(model.summary())
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
return model



from sklearn.model_selection import KFold #
evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=2):
 scores, histories = list(), list()
# prepare cross validation
 kfold = KFold(n_folds, shuffle=True, random_state=1)
 # enumerate splits  for train_ix, test_ix in
kfold.split(dataX):
 # define model   model = build_model()  # select rows for train and test   trainX, trainY,
testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
 # fit model   history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX,
testY), verbose=0)  # evaluate model
  _, acc = model.evaluate(testX, testY, verbose=0)
print('> %.3f' % (acc * 100.0))
  # stores scores
scores.append(acc)
histories.append(history)
return scores, histories


# plot diagnostic learning curves def
summarize_diagnostics(histories):
for i in range(len(histories)):
 # plot loss   plt.subplot(2, 1,
1)   plt.title('Cross Entropy
Loss')
plt.plot(histories[i].history['l
oss'], color='blue',
label='train')
plt.plot(histories[i].history['v
al_loss'], color='orange',
label='test')
 # plot accuracy   plt.subplot(2, 1, 2)   plt.title('Classification Accuracy')
plt.plot(histories[i].history['accuracy'], color='blue', label='train')
```

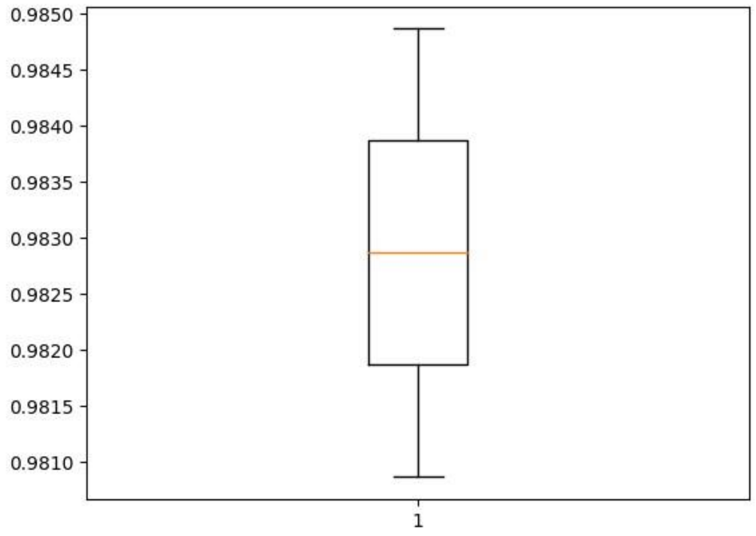
```
plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
plt.show()


# summarize model performance def
summarize_performance(scores):
 # print summary  print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100,
std(scores)*100, len(scores)))
 # box and whisker plots of results
plt.boxplot(scores)  plt.show()


# run the test harness for evaluating a model
def run_test_harness():  # load dataset
trainX, trainY, testX, testY = load_dataset()
 # prepare pixel data
 trainX, testX = prep_pixels(trainX, testX)
 # evaluate model  scores, histories =
evaluate_model(trainX, trainY)
 # learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)


# entry point, run the test harness
run_test_harness()
```

```
 dropout_14 (Dropout)        (None, 32)                0
dense_9 (Dense)             (None, 10)                330
=================================================================
Total params: 216426 (845.41 KB)
Trainable params: 216426 (845.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
> 98.487
```






```
Accuracy: mean=98.287 std=0.200, n=2
```



Ex 14-Implementing

GAN Aim: To implement GAN Algorithm:

1. Importing all libraries
2. Getting the Dataset
3. Data Preparation – It includes various steps to accomplish like preprocessing data, scaling, flattening, and reshaping the data.
4. Define the function Generator and Discriminator.
5. Create a Random Noise and then create an Image with Random Noise.
6. Setting Parameters like defining epoch, batch size, and Sample size.
7. Define the function of generating Sample Images.
8. Train Discriminator then trains Generator and it will create Images.
9. Will see what clarity of Images is created by Generator.

```
import numpy as np import pandas as pd import matplotlib.pyplot as plt import os import
tensorflow as tf from tensorflow.keras.layers import Input, Dense, LeakyReLU, Dropout,
BatchNormalization from tensorflow.keras.models import Model from
tensorflow.keras.optimizers import SGD, Adam
```

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data() # Scale
the inputs in range of (-1, +1) for better training x_train,
x_test = x_train / 255.0 * 2 - 1, x_test / 255.0 * 2 - 1
```

```
for i in range(49):
plt.subplot(7, 7, i+1)
plt.axis("off")   #plot
raw pixel data
plt.imshow(x_train[i])
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11490434/11490434 [==============================] - 0s 0us/step
```

Flattening and Scaling the data As the dimension of the dataset is 3 so we will atten it to 2 dimensions and 28*28 means 684 and get converted to 60000 by 684.

```
N, H, W = x_train.shape  #number, height, width
D = H * W   #dimension (28, 28) x_train =
x_train.reshape(-1, D) x_test =
x_test.reshape(-1, D)
```

```
# Defining Generator Model latent_dim = 100 def build_generator(latent_dim):
i = Input(shape=(latent_dim,))   x = Dense(256,
activation=LeakyReLU(alpha=0.2))(i)   x = BatchNormalization(momentum=0.7)(x)
x = Dense(512, activation=LeakyReLU(alpha=0.2))(x)   x =
BatchNormalization(momentum=0.7)(x)   x = Dense(1024,
activation=LeakyReLU(alpha=0.2))(x)   x = BatchNormalization(momentum=0.7)(x)
x = Dense(D, activation='tanh')(x)  #because Image pixel is between -1 to 1.
  model = Model(i, x)  #i is input x is output layer
return model
```

De ning Discriminator Model Here we develop a simple Feed Forward Neural network for Discriminator where we will pass an image size. The activation function used is Leaky ReLU and you know the reason for it and sigmoid is used in the output layer for binary classi cation problems to classify Images as real or Fake.

```
def build_discriminator(img_size):
  i = Input(shape=(img_size,))   x = Dense(512,
activation=LeakyReLU(alpha=0.2))(i)   x =
Dense(256, activation=LeakyReLU(alpha=0.2))(x)   x
= Dense(1, activation='sigmoid')(x)   model =
Model(i, x)   return model
```

Compile Models Now it's time to compile both the de ned components of GANs

```
# Build and compile the discriminator discriminator = build_discriminator(D) discriminator.compile (
loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
# Build and compile the combined model
generator = build_generator(latent_dim)
```

```
## Create an input to represent noise sample from latent space
z = Input(shape=(latent_dim,)) ## Pass noise through a
generator to get an Image img = generator(z)
discriminator.trainable = False fake_pred =
discriminator(img)
```

Create Generator Model It's time to create a combined Generator model with noise input and feedback of discriminator that helps the generator to improve its performance.

```
combined_model_gen = Model(z, fake_pred)  #first is noise and 2nd is fake prediction
# Compile the combined model
combined_model_gen.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
```

De ning Parameters for the training of GAN De ne epochs, batch size, and a sample period which means after how many steps the generator will create a sample. After this, we de ne the Batch labels as one and zero. One represents that image is real and zero represents the image is fake. And we also create two empty lists to store the loss of generator and discriminator. And very importantly we create an empty le in the working directory where the generated image through the generator will be saved.

```
batch_size = 32 epochs =
12000 sample_period = 200
ones = np.ones(batch_size)
zeros =
np.zeros(batch_size)
#store generator and discriminator loss in each step or each epoch
d_losses = [] g_losses = []
#create a file in which generator will create and save images
if not os.path.exists('gan_images'):
os.makedirs('gan_images')
```

Function to create Sample Images Create a function that generates a grid of random samples from a generator and saves them to a le. In simple words, it will create random images on some epochs. We de ne the row size as 5 and column as also 5 so in a single iteration or on a single page it will generate 25 images.

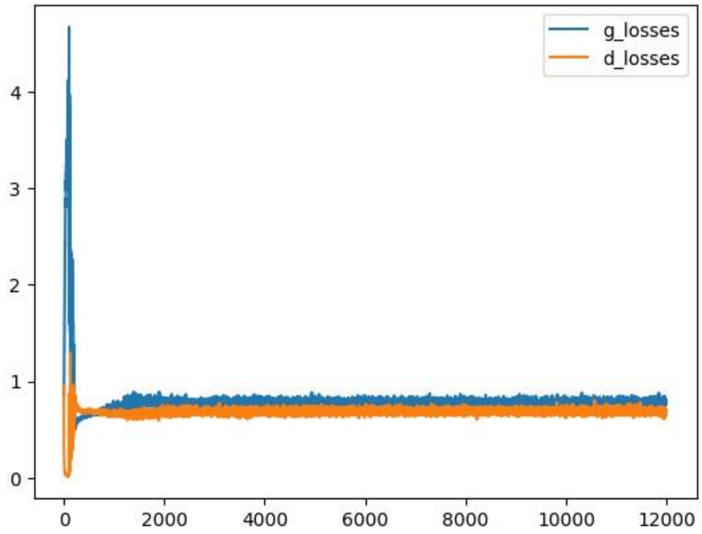
```
def sample_images(epoch):
  rows, cols = 5, 5   noise = np.random.randn(rows
* cols, latent_dim)   imgs =
generator.predict(noise)
  # Rescale images 0 - 1   imgs = 0.5 * imgs + 0.5   fig, axs =
plt.subplots(rows, cols)  #fig to plot img and axis to store   idx = 0
for i in range(rows):  #5*5 loop means on page 25 imgs will be there
for j in range(cols):       axs[i,j].imshow(imgs[idx].reshape(H, W),
cmap='gray')       axs[i,j].axis('off')       idx += 1
fig.savefig("gan_images/%d.png" % epoch)   plt.close()
#FIRST we will train Discriminator(with real imgs and fake imgs)
# Main training loop for
epoch in range(epochs):
###########################
  ### Train discriminator ###
  ###########################   #
Select a random batch of images
  idx = np.random.randint(0, x_train.shape[0], batch_size)
real_imgs = x_train[idx]  #MNIST dataset
  # Generate fake images   noise = np.random.randn(batch_size, latent_dim)
#generator to generate fake imgs   fake_imgs = generator.predict(noise)   # Train
the discriminator
```

```
  # both loss and accuracy are returned   d_loss_real, d_acc_real = discriminator.train_on_batch(real_imgs,
ones)  #belong to positive class(real imgs)   d_loss_fake, d_acc_fake =
discriminator.train_on_batch(fake_imgs, zeros)  #fake imgs   d_loss = 0.5 * (d_loss_real + d_loss_fake)
d_acc  = 0.5 * (d_acc_real + d_acc_fake)
  #######################
  ### Train generator ###   #######################
noise = np.random.randn(batch_size, latent_dim)   g_loss
= combined_model_gen.train_on_batch(noise, ones)
  #Now we are trying to fool the discriminator that generate imgs are real that's why we are providing label as 1
  # do it again!   noise = np.random.randn(batch_size,
latent_dim)   g_loss =
combined_model_gen.train_on_batch(noise, ones)
  # Save the losses   d_losses.append(d_loss)  #save
the loss at each epoch   g_losses.append(g_loss)   if
epoch % 100 == 0:
    print("epoch: {epoch+1}/{epochs}, d_loss: {d_loss:.2f}, d_acc: {d_acc:.2f}, g_loss: {g_loss:.2f}")
if epoch % sample_period == 0:
    sample_images(epoch)
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
```

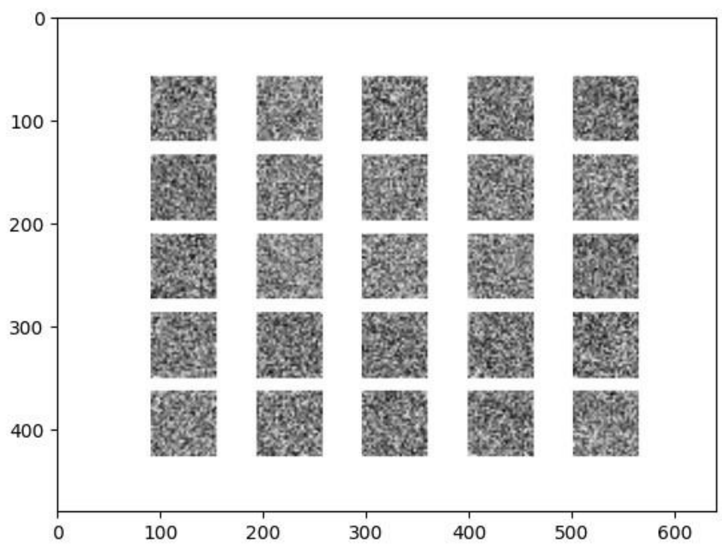
```
plt.plot(g_losses, label='g_losses')
plt.plot(d_losses, label='d_losses')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7cd379032860>
```



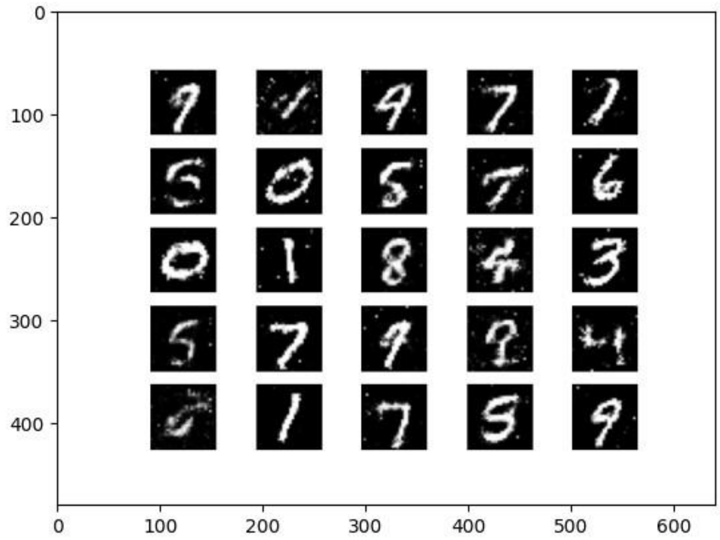

```
#Plot the generated Image at zero epoch
from skimage.io import imread
a = imread('gan_images/0.png')
plt.imshow(a)
```

```
<matplotlib.image.AxesImage at 0x7cd330e15bd0>
```




```
from skimage.io import imread
a = imread('gan_images/10000.png')
plt.imshow(a)
```

```
<matplotlib.image.AxesImage at 0x7cd330eb78e0>
```



0s completed at 4:24 PM