NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA-201306

Department of CSE (AIML)

Session (2023 – 2024)

LAB FILE

Deep Learning Lab

(ACSML0652)

(6th Semester)

Submitted To:
Mr. Subhash Chandra

Submitted By:
ANMOL R SRIVASTAVA
2101331530024
AIML- A

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute)

School of Computer Sciences & Engineering in Emerging Technologies

Deep Learning Lab (ACSML0652)

INDEX

Experiment 1

Write a program Print Dimensions of dataset

Code:

```
import pandas as pd

df = pd.read_csv("/content/sample_data/mnist_test.csv")

print(df.head())                          #returns top 5 rows/tuples

print("Shape of the dataset",df.shape)    #returns the shape (dimensions) of the dataset

print("Size of the dataset",df.size)      #returns the total number of cells
```

Output:

	7	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.658	0.659	0.660	0.661	0.662	0.663	0.664	0.665	0.666	0.667
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

```
Shape of the dataset (9999, 785)
Size of the dataset 7849215
```

Experiment 2

Write a program to Calculate of Accuracy Values.

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Loading the dataset
X, Y = load_iris(return_X_y = True)

# Splitting the dataset in training and test data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0)

# Training the model using the Support Vector Classification class of sklearn
svc = SVC()
svc.fit(X_train, Y_train)

# Computing the accuracy score of the model
Y_pred = svc.predict(X_test)
score = accuracy_score(Y_test, Y_pred)
print("Accuracy Score :",score)
```

Output:

```
Accuracy Score : 0.9777777777777777
```

Experiment 3

Write a program to Build an Artificial Neural Network Classifier

Code:

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = load_iris()
X,y = iris.data, iris.target
#preprocess the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
#split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state = 42)
# Build the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10,activation ='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(1,activation='sigmoid')])
model.compile(optimizer ='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X_train,y_train,epochs=50,batch_size=32,validation_split=0.2)
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
#Calculate accuracy
acc = accuracy_score(y_test, y_pred_binary)
print(acc)
```

Output:

```
0.6333333333333333
```

Experiment 4

Write a program to Compose Matrix Shape and Tensor Shape

Code:

```
import tensorflow as tf

def compose_matrix_shape(matrix_shape):
    return tuple(matrix_shape)

def compose_tensor_shape(tensor_shape):
    return tf.TensorShape(tensor_shape)

# Compose matrix shape
matrix_shape = [3, 4]
composed_matrix_shape = compose_matrix_shape(matrix_shape)
print("Composed Matrix Shape:", composed_matrix_shape)

# Compose tensor shape
tensor_shape = [None, 5, 5]
composed_tensor_shape = compose_tensor_shape(tensor_shape)
print("Composed Tensor Shape:", composed_tensor_shape)
```

Output:

```
Composed Matrix Shape: (3, 4)
Composed Tensor Shape: (None, 5, 5)
```

Experiment 5

Write a program to showing accessing and manipulation of tensors.

Code:

```
import tensorflow as tf

# Create a tensor
tensor = tf.constant([[1, 2, 3], [4, 5, 6]])

# Accessing elements of the tensor
print("Tensor:")
print(tensor)
print("Shape:", tensor.shape)
print("Data type:", tensor.dtype)
print("Number of dimensions:", tensor.ndim)

# Accessing specific elements
print("Accessing specific elements:")
print("Element at (0, 0):", tensor[0, 0].numpy())
print("Element at (1, 2):", tensor[1, 2].numpy())

# Manipulating tensors
# Addition
tensor_add = tensor + 10
print("Tensor + 10:")
print(tensor_add)

# Multiplication
tensor_mul = tensor * 2
print("Tensor * 2:")
print(tensor_mul)

# Reshaping tensor
tensor_reshaped = tf.reshape(tensor, (3, 2))
print("Reshaped tensor (3x2):")
print(tensor_reshaped)
```

Output:

 # Accessing elements of the tensor

```
Tensor:
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
Shape: (2, 3)
Data type: <dtype: 'int32'>
Number of dimensions: 2
```

 # Accessing specific elements

```
Accessing specific elements:
Element at (0, 0): 1
Element at (1, 2): 6
```

 # Manipulating tensors – Addition

```
Manipulating tensors:
Tensor + 10:
tf.Tensor(
[[11 12 13]
 [14 15 16]], shape=(2, 3), dtype=int32)
```

 # Multiplication

```
Tensor * 2:
tf.Tensor(
[[ 2  4  6]
 [ 8 10 12]], shape=(2, 3), dtype=int32)
```

 # Reshaping tensor

```
Reshaped tensor (3x2):
tf.Tensor(
[[1 2]
 [3 4]
 [5 6]], shape=(3, 2), dtype=int32)
```

Experiment 6

Write a program to understand the mechanism of practically training a binary classifier

Code:

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Split the data 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)

# Define the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

Output:

```
Epoch 1/10
25/25 [==============================] - 1s 11ms/step - loss: 0.6588 - accuracy: 0.5850 - val_loss: 0.5801 - val_accuracy: 0.7600
Epoch 2/10
25/25 [==============================] - 0s 3ms/step - loss: 0.5056 - accuracy: 0.8087 - val_loss: 0.4766 - val_accuracy: 0.8150
Epoch 3/10
25/25 [==============================] - 0s 3ms/step - loss: 0.4061 - accuracy: 0.8438 - val_loss: 0.4135 - val_accuracy: 0.8300
Epoch 4/10
25/25 [==============================] - 0s 3ms/step - loss: 0.3475 - accuracy: 0.8650 - val_loss: 0.3923 - val_accuracy: 0.8450
Epoch 5/10
25/25 [==============================] - 0s 3ms/step - loss: 0.3183 - accuracy: 0.8750 - val_loss: 0.3801 - val_accuracy: 0.8550
Epoch 6/10
25/25 [==============================] - 0s 3ms/step - loss: 0.3006 - accuracy: 0.8850 - val_loss: 0.3776 - val_accuracy: 0.8550
Epoch 7/10
25/25 [==============================] - 0s 3ms/step - loss: 0.2900 - accuracy: 0.8863 - val_loss: 0.3716 - val_accuracy: 0.8500
Epoch 8/10
25/25 [==============================] - 0s 4ms/step - loss: 0.2769 - accuracy: 0.8950 - val_loss: 0.3781 - val_accuracy: 0.8550
Epoch 9/10
25/25 [==============================] - 0s 3ms/step - loss: 0.2688 - accuracy: 0.8900 - val_loss: 0.3728 - val_accuracy: 0.8500
Epoch 10/10
25/25 [==============================] - 0s 3ms/step - loss: 0.2599 - accuracy: 0.8988 - val_loss: 0.3789 - val_accuracy: 0.8500
7/7 [==============================] - 0s 2ms/step - loss: 0.3789 - accuracy: 0.8500
Test Accuracy: 0.8500000238418579
```

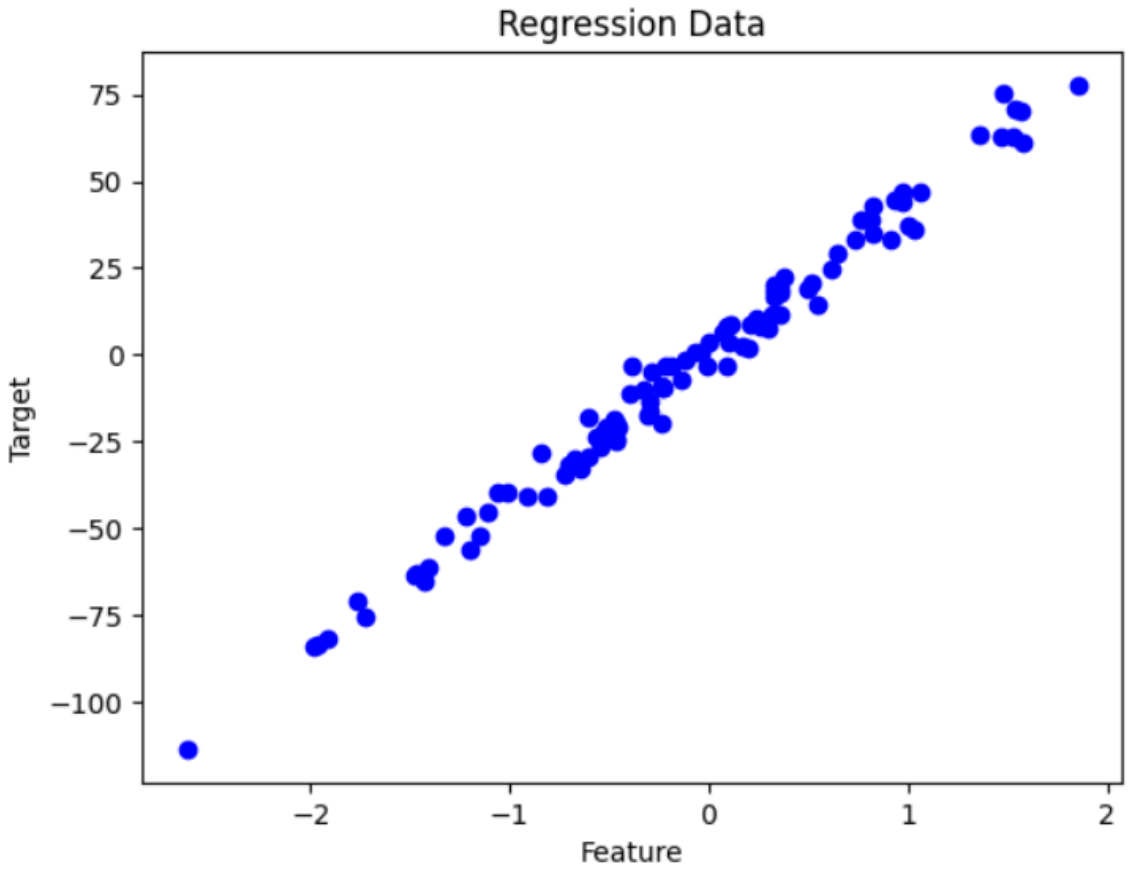
Experiment 7

Write a program to show regression Data sampling.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression

# Generate synthetic regression data
X, y = make_regression(n_samples=100, n_features=1, noise=5, random_state=42)

# Plot the data
plt.scatter(X, y, color='blue')
plt.title('Regression Data')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.show()
```

Output:



Experiment 8

Write a program to Combat Overfitting

Code:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Split the data 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)

# Define a function to create a model
def create_model(input_dim, regularizer=None, dropout_rate=None):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=input_dim, kernel_regularizer=regularizer))
    if dropout_rate:
        model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu', kernel_regularizer=regularizer))
    if dropout_rate:
        model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    return model

# Baseline model
print("Baseline Model:")
baseline_model = create_model(X_train.shape[1])
baseline_model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
baseline_history = baseline_model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))

# Regularization - L2
print("\nModel with L2 Regularization:")
```

```
l2_model = create_model(X_train.shape[1], regularizer=l2(0.001))
l2_model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
l2_history = l2_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Dropout
print("\nModel with Dropout:")
dropout_model = create_model(X_train.shape[1], dropout_rate=0.2)
dropout_model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
dropout_history = dropout_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test))
```

Output:

Baseline model

```
Epoch 50/50
25/25 [==============================] - 0s 3ms/step - loss: 0.0475 - accuracy: 0.9962 - val_loss: 0.5421 - val_accuracy: 0.8400
```

Regularization - L2

```
Epoch 50/50
25/25 [==============================] - 0s 3ms/step - loss: 0.1331 - accuracy: 0.9887 - val_loss: 0.5941 - val_accuracy: 0.8100
```

Dropout

```
Epoch 50/50
25/25 [==============================] - 0s 3ms/step - loss: 0.1680 - accuracy: 0.9350 - val_loss: 0.4122 - val_accuracy: 0.8300
```

Experiment 9

Write a program to build a simple autoencoder based on a fully connected layer in keras.

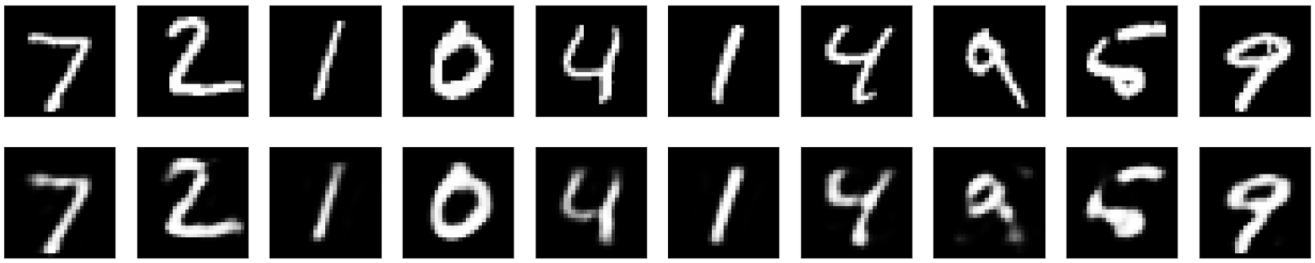
Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess the dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.reshape(x_train, (len(x_train), np.prod(x_train.shape[1:])))
x_test = np.reshape(x_test, (len(x_test), np.prod(x_test.shape[1:])))

# Define the autoencoder architecture
input_dim = x_train.shape[1]
encoding_dim = 32

input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
# Encode and decode some digits
encoded_imgs = autoencoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

# Plot original and reconstructed images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output:

```
Epoch 50/50
235/235 [==============================] - 2s 8ms/step - loss: 0.0926 - val_loss: 0.0915
313/313 [==============================] - 1s 1ms/step
313/313 [==============================] - 0s 1ms/step
```



Experiment 10

Write a program to Build an Convolutional Neural Network

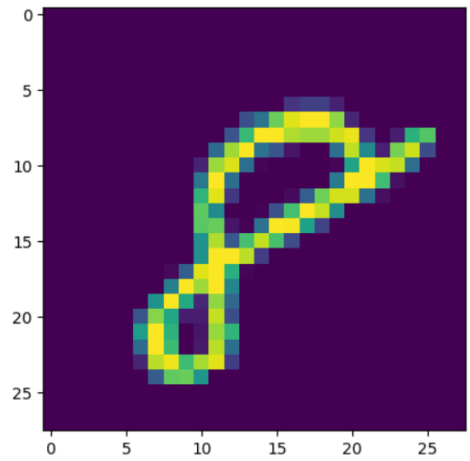
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras import models, layers, optimizers, losses, metrics
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation = 'relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()

#Load dataset and preprocess data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
plt.imshow(train_images[59999])
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)



#compile and train model
model.compile(
    optimizer='rmsprop',
```

```
    loss='categorical_crossentropy',
    metrics = ['accuracy']
)
model.fit(train_images, train_labels, epochs=5, batch_size=64)
model.evaluate(test_images, test_labels)
```

Output:

```
<matplotlib.image.AxesImage at 0x7b5736081c30>
```



Experiment 11

Program for Multi-Classification using MNIST Dataset

Code:

```
import tensorflow as tf
import keras
from keras import Sequential
from keras.layers import Dense, Flatten
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)


import matplotlib.pyplot as plt
plt.imshow(x_train[4])

x_train = x_train/255
x_test = x_test/255
model=Sequential()
model.add(Flatten(input_shape = (28,28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.summary()
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam', metrics = ['accuracy'])
history = model.fit(x_train, y_train, epochs=25, validation_split = 0.2)
y_prob = model.predict(x_test)
y_pred = y_prob.argmax(axis = 1)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Output:

```
0.9791
```

Experiment 12

Write a program to Build Cat vs Dog prediction model using transfer learning

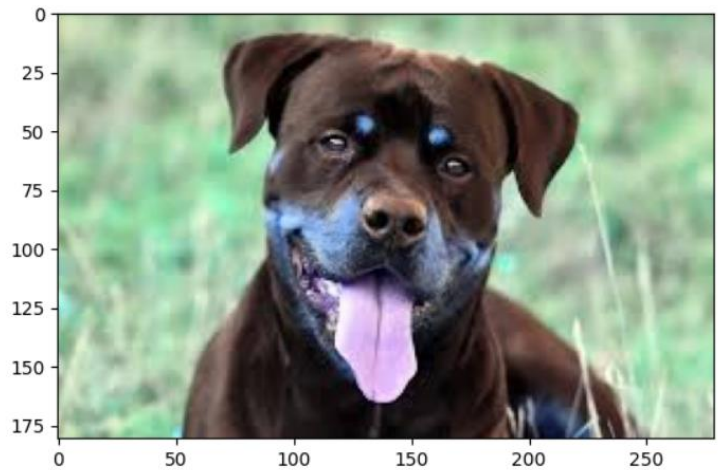
Code:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d salader/dogs-vs-cats
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
import tensorflow as tf
import keras
from keras import Sequential
from keras.layers import Dense, Flatten
from keras.applications.vgg16 import VGG16
conv_base = VGG16(
    weights = 'imagenet',
    include_top = False,
    input_shape = (150, 150, 3)
)
conv_base.summary()

model=Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.summary()

conv_base.trainable = False
model.summary()
# generators
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels = 'inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(150,150)
)


validation_ds = keras.utils.image_dataset_from_directory(
```

```
    directory = '/content/test',
    labels = 'inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(150,150)
)

#Normalize
def process(image,label):
  image = tf.cast(image/255, tf.float32)
  return image, label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history=model.fit(train_ds,epochs=10, validation_data=validation_ds)
import matplotlib.pyplot as plt
import cv2
import cv2 as cv
import numpy as np
import argparse
import time
import cv2
test_img = cv2.imread('/content/Dog.png')
plt.imshow(test_img)
Test_img.shape
test_img = cv2.resize(test_img, (150, 150))
test_input = test_img.reshape((1,150,150,3))
model.predict(test_input)
```

Output:

```
<matplotlib.image.AxesImage at 0x7b73f8de7580>
```




```
1/1 [==============================] - 1s 1s/step
array([[1.]], dtype=float32)
```

Experiment 13

Write a program to Build Long Short Term Memory

Code:

```
faqs = """About the Program
What is the course fee for  Data Science Mentorship Program (DSMP 2023)
The course follows a monthly subscription model where you have to make monthly payments of Rs 799/month.
What is the total duration of the course?
The total duration of the course is 7 months. So the total course fee becomes 799*7 = Rs 5600(approx.)
What is the syllabus of the mentorship program?
We will be covering the following modules:
Python Fundamentals
Python libraries for Data Science
Data Analysis
SQL for Data Science
Maths for Machine Learning
ML Algorithms
Practical ML
MLOPs
Case studies
You can check the detailed syllabus here - https://learnwith.campusx.in/courses/CampusX-Data-Science-Mentorship-Program-637339afe4b0615a1bbed390
Will Deep Learning and NLP be a part of this program?
No, NLP and Deep Learning both are not a part of this program's curriculum.
What if I miss a live session? Will I get a recording of the session?
Yes all our sessions are recorded, so even if you miss a session you can go back and watch the recording.
Where can I find the class schedule?
Checkout this google sheet to see month by month time table of the course - https://docs.google.com/spreadsheets/d/16OoTax_A6ORAeCg4emgexhqqPv3noQPYKU7RJ6ArOzk/edit?usp=sharing.
What is the time duration of all the live sessions?
Roughly, all the sessions last 2 hours.
What is the language spoken by the instructor during the sessions?
Hinglish
How will I be informed about the upcoming class?
You will get a mail from our side before every paid session once you become a paid user.
Can I do this course if I am from a non-tech background?
Yes, absolutely.
I am late, can I join the program in the middle?
Absolutely, you can join the program anytime.
If I join/pay in the middle, will I be able to see all the past lectures?
Yes, once you make the payment you will be able to see all the past content in your dashboard.
Where do I have to submit the task?
```

You don’t have to submit the task. We will provide you with the solutions, you have to self evaluate the task yourself.
Will we do case studies in the program?
Yes.
Where can we contact you?
You can mail us at nitish.campusx@gmail.com
Payment/Registration related questions
Where do we have to make our payments? Your YouTube channel or website?
You have to make all your monthly payments on our website. Here is the link for our website - https://learnwith.campusx.in/
Can we pay the entire amount of Rs 5600 all at once?
Unfortunately no, the program follows a monthly subscription model.
What is the validity of monthly subscription? Suppose if I pay on 15th Jan, then do I have to pay again on 1st Feb or 15th Feb
15th Feb. The validity period is 30 days from the day you make the payment. So essentially you can join anytime you don’t have to wait for a month to end.
What if I don’t like the course after making the payment. What is the refund policy?
You get a 7 days refund period from the day you have made the payment.
I am living outside India and I am not able to make the payment on the website, what should I do?
You have to contact us by sending a mail at nitish.campusx@gmail.com
Post registration queries
Till when can I view the paid videos on the website?
This one is tricky, so read carefully. You can watch the videos till your subscription is valid. Suppose you have purchased subscription on 21st Jan, you will be able to watch all the past paid sessions in the period of 21st Jan to 20th Feb. But after 21st Feb you will have to purchase the subscription again.
But once the course is over and you have paid us Rs 5600(or 7 installments of Rs 799) you will be able to watch the paid sessions till Aug 2024.
Why lifetime validity is not provided?
Because of the low course fee.
Where can I reach out in case of a doubt after the session?
You will have to fill a google form provided in your dashboard and our team will contact you for a 1 on 1 doubt clearance session
If I join the program late, can I still ask past week doubts?
Yes, just select past week doubt in the doubt clearance google form.
I am living outside India and I am not able to make the payment on the website, what should I do?
You have to contact us by sending a mail at nitish.campusx@gmai.com
Certificate and Placement Assistance related queries
What is the criteria to get the certificate?
There are 2 criterias:
You have to pay the entire fee of Rs 5600
You have to attempt all the course assessments.
I am joining late. How can I pay payment of the earlier months?
You will get a link to pay fee of earlier months in your dashboard once you pay for the current month.
I have read that Placement assistance is a part of this program. What comes under Placement assistance?
This is to clarify that Placement assistance does not mean Placement guarantee. So we dont guarantee you any jobs or for that matter even interview calls. So if you are planning to join this course just for placements, I am afraid you will be disappointed. Here is what comes under placement assistance
Portfolio Building sessions
Soft skill sessions
Sessions with industry mentors
Discussion on Job hunting strategies

```
"""

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts([faqs])
len(tokenizer.word_index)
input_sequences = []
for sentence in faqs.split('\n'):
  tokenized_sentence = tokenizer.texts_to_sequences([sentence])[0]

  for i in range(1,len(tokenized_sentence)):
    input_sequences.append(tokenized_sentence[:i+1])
print(input_sequences)



max_len = max([len(x) for x in input_sequences])
print(max_len)

from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_input_sequences = pad_sequences(input_sequences, maxlen = max_len, padding='pre')
print(padded_input_sequences)

x = padded_input_sequences[:,:-1]
print(x)

y = padded_input_sequences[:,-1]
print(y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
model = Sequential()
model.add(Embedding(283, 100, input_length = 56))
model.add(LSTM(150))

model.add(Dense(283, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

from tensorflow.keras.utils import to_categorical
y = to_categorical(y, num_classes = 283)
print(y.shape)
```

```
(863, 283)
```

```
model.fit(x,y,epochs = 100)

import numpy as np
import time
text = "No, NLP and Deep"
```

```
for i in range(10):
  #tokenize
  token_text = tokenizer.texts_to_sequences([text])[0]
  #padding
  padded_token_text = pad_sequences([token_text],maxlen=56, padding='pre')
  #predict
  pos = np.argmax(model.predict(padded_token_text))



  for word,index in  tokenizer.word_index.items():
    if index == pos:
      text = text + " " + word
      print(text)
      time.sleep(2)
```

Output:

```
1/1 [==============================] - 0s 53ms/step
No, NLP and Deep learning
1/1 [==============================] - 0s 33ms/step
No, NLP and Deep learning both
1/1 [==============================] - 0s 31ms/step
No, NLP and Deep learning both are
1/1 [==============================] - 0s 28ms/step
No, NLP and Deep learning both are not
1/1 [==============================] - 0s 29ms/step
No, NLP and Deep learning both are not a
1/1 [==============================] - 0s 47ms/step
No, NLP and Deep learning both are not a part
1/1 [==============================] - 0s 33ms/step
No, NLP and Deep learning both are not a part of
1/1 [==============================] - 0s 31ms/step
No, NLP and Deep learning both are not a part of this
1/1 [==============================] - 0s 29ms/step
No, NLP and Deep learning both are not a part of this program’s
1/1 [==============================] - 0s 28ms/step
No, NLP and Deep learning both are not a part of this program’s curriculum
```

Experiment 14

Write a program Integer Encoding Using Simple RNN.

Code:

```
import numpy as np
docs = [
    "go india",
    "india india","hip hip hurray","jeetega bhai jeetega india jeetega","bharat mata ki jai","kholi kholi",
    "sachin sachin","dhoni dhoni","modi ji ki jai","inqualab jindabad"
]
from keras.preprocessing.text import Tokenizer
tokenizer= Tokenizer(oov_token="<nothing>")
tokenizer.fit_on_texts(docs)
tokenizer.word_index

tokenizer.word_counts

tokenizer.document_count

sequences = tokenizer.texts_to_sequences(docs)
sequences

from keras.utils import pad_sequences
sequences = pad_sequences(sequences, padding='post')
sequences

from keras.datasets import imdb
from keras import Sequential
from keras.layers import Dense, SimpleRNN, Embedding, Flatten

(x_train, y_train),(x_test,y_test) = imdb.load_data()
(x_train, y_train),(x_test,y_test) = imdb.load_data()

len(x_train[2])

x_train = pad_sequences(x_train,padding='post', maxlen=50)
x_test = pad_sequences(x_test,padding='post', maxlen=50)
```

```
x_train[0]

model = Sequential()

model.add(SimpleRNN(32, input_shape=(50,1),return_sequences=False))
model.add(Dense(1,activation='sigmoid'))

model.summary()

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train,y_train,epochs=5,validation_data=(x_test,y_test))
```

Output:

```
{'<nothing>': 1,
 'india': 2,
 'jeetega': 3,
 'hip': 4,
 'ki': 5,
 'jai': 6,
 'kholi': 7,
 'sachin': 8,
 'dhoni': 9,
 'go': 10,
 'hurray': 11,
 'bhai': 12,
 'bharat': 13,
 'mata': 14,
 'modi': 15,
 'ji': 16,
 'inqualab': 17,
 'jindabad': 18}
```

```
OrderedDict([('go', 1),
             ('india', 4),
             ('hip', 2),
             ('hurray', 1),
             ('jeetega', 3),
             ('bhai', 1),
             ('bharat', 1),
             ('mata', 1),
             ('ki', 2),
             ('jai', 2),
             ('kholi', 2),
             ('sachin', 2),
             ('dhoni', 2),
             ('modi', 1),
             ('ji', 1),
             ('inqualab', 1),
             ('jindabad', 1)])
```

```
array([[10,  2,  0,  0,  0],
       [ 2,  2,  0,  0,  0],
       [ 4,  4, 11,  0,  0],
       [ 3, 12,  3,  2,  3],
       [13, 14,  5,  6,  0],
       [ 7,  7,  0,  0,  0],
       [ 8,  8,  0,  0,  0],
       [ 9,  9,  0,  0,  0],
       [15, 16,  5,  6,  0],
       [17, 18,  0,  0,  0]], dtype=int32)
```

```
array([2071,   56,   26,  141,    6,  194, 7486,   18,    4,  226,   22,
         21,  134,  476,   26,  480,    5,  144,   30, 5535,   18,   51,
         36,   28,  224,   92,   25,  104,    4,  226,   65,   16,   38,
       1334,   88,   12,   16,  283,    5,   16, 4472,  113,  103,   32,
         15,   16, 5345,   19,  178,   32], dtype=int32)
```

```
Epoch 1/5
782/782 [==============================] - 15s 16ms/step - loss: 0.6943 - accuracy: 0.5106 - val_loss: 0.6946 - val_accuracy: 0.5037
Epoch 2/5
782/782 [==============================] - 13s 17ms/step - loss: 0.6928 - accuracy: 0.5106 - val_loss: 0.6940 - val_accuracy: 0.5042
Epoch 3/5
782/782 [==============================] - 12s 15ms/step - loss: 0.6926 - accuracy: 0.5164 - val_loss: 0.6958 - val_accuracy: 0.5067
Epoch 4/5
782/782 [==============================] - 11s 14ms/step - loss: 0.6929 - accuracy: 0.5086 - val_loss: 0.6935 - val_accuracy: 0.5052
Epoch 5/5
782/782 [==============================] - 12s 15ms/step - loss: 0.6930 - accuracy: 0.5080 - val_loss: 0.6954 - val_accuracy: 0.5059
<keras.src.callbacks.History at 0x7840b1454910>
```

Experiment 15

Write a program to Build Embedding Sentiment Analysis Using Simple RNN.

Code:

```
docs = [
    "go india",
    "india india","hip hip hurray","jeetega bhai jeetega india jeetega","bharat mata ki jai","kholi kholi",
    "sachin sachin","dhoni dhoni","modi ji ki jai","inqualab jindabad"
]

from keras.datasets import imdb
from keras import Sequential
from keras.layers import Dense, SimpleRNN, Embedding, Flatten
from keras.preprocessing.text import Tokenizer
tokenizer= Tokenizer(oov_token="<nothing>")
tokenizer.fit_on_texts(docs)
tokenizer.word_index
```

```
{'<nothing>': 1,
 'india': 2,
 'jeetega': 3,
 'hip': 4,
 'ki': 5,
 'jai': 6,
 'kholi': 7,
 'sachin': 8,
 'dhoni': 9,
 'go': 10,
 'hurray': 11,
 'bhai': 12,
 'bharat': 13,
 'mata': 14,
 'modi': 15,
 'ji': 16,
 'inqualab': 17,
 'jindabad': 18}
```

```
tokenizer.word_counts
```

```
OrderedDict([('go', 1),
             ('india', 4),
             ('hip', 2),
             ('hurray', 1),
             ('jeetega', 3),
             ('bhai', 1),
             ('bharat', 1),
             ('mata', 1),
             ('ki', 2),
             ('jai', 2),
             ('kholi', 2),
             ('sachin', 2),
             ('dhoni', 2),
             ('modi', 1),
             ('ji', 1),
             ('inqualab', 1),
             ('jindabad', 1)])
```

tokenizer.document_count

```
10
```

sequences = tokenizer.texts_to_sequences(docs)
sequences

```
[[10, 2],
 [2, 2],
 [4, 4, 11],
 [3, 12, 3, 2, 3],
 [13, 14, 5, 6],
 [7, 7],
 [8, 8],
 [9, 9],
 [15, 16, 5, 6],
 [17, 18]]
```

(x_train, y_train), (x_test, y_test) = imdb.load_data()
from keras.utils import pad_sequences
x_train = pad_sequences(x_train, padding='post',maxlen=50)
x_test = pad_sequences(x_test,padding='post', maxlen=50)
x_train.shape

```
(25000, 50)
```

model = Sequential()

model.add(Embedding(100000, output_dim=2, input_length=50))
model.add(SimpleRNN(32, return_sequences=False))
model.add(Dense(1,activation='sigmoid'))

model.summary()

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 50, 2)             200000

 simple_rnn_4 (SimpleRNN)    (None, 32)                1120

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 201153 (785.75 KB)
Trainable params: 201153 (785.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

```
Epoch 1/5
782/782 [==============================] - 19s 23ms/step - loss: 0.6371 - acc: 0.6038 - val_loss: 0.5179 - val_acc: 0.7632
Epoch 2/5
782/782 [==============================] - 18s 23ms/step - loss: 0.3819 - acc: 0.8331 - val_loss: 0.4198 - val_acc: 0.8094
Epoch 3/5
782/782 [==============================] - 15s 20ms/step - loss: 0.2552 - acc: 0.9002 - val_loss: 0.4488 - val_acc: 0.8044
Epoch 4/5
782/782 [==============================] - 15s 20ms/step - loss: 0.1836 - acc: 0.9353 - val_loss: 0.4991 - val_acc: 0.7974
Epoch 5/5
782/782 [==============================] - 18s 23ms/step - loss: 0.1364 - acc: 0.9537 - val_loss: 0.5764 - val_acc: 0.7879
<keras.src.callbacks.History at 0x7b79da785b40>
```

Experiment 16

Write a program for Logistic regression model (Spam-ham)

Code:

```
# Reading Data
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/mohitgupta-omg/Kaggle-SMS-Spam-Collection-Dataset-/master/spam.csv', encoding='latin-1')
data.head()
data.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1, inplace=True)
data.columns = ['label', 'text']
data.head()
data.isna().sum()
import nltk
nltk.download('all')
text = list(data['text'])
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
lematizer= WordNetLemmatizer()
corpus = []
for i in range(len(text)):
  r = re.sub('[^a=zA-Z]',' ',text[i])
  r = r.lower()
  r = r.split()
  r = [word for word in r if word not in stopwords.words('english')]
  r = ' '.join(r)
  corpus.append(r)
data['text'] = corpus
data.head()

X=data['text']
y=data['label']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=123)
print("Training Data:",X_train.shape)
print("Testing Data:",X_test.shape)

from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer()
X_train_cv=cv.fit_transform(X_train)
X_train_cv.shape

#Training Logistic Regressio Model
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train_cv, y_train)
X_test_cv = cv.transform(X_test)
predictions = lr.predict(X_test_cv)
predictions

import pandas as pd
from sklearn import metrics

df = pd.DataFrame(metrics.confusion_matrix(y_test,predictions), index=['ham','spam'], columns=['ham','spam'])
```

Output:

	ham	spam
ham	1600	2
spam	31	206

Experiment 17

Write a program for ANN classification.

```
# Reading the cleaned numeric titanic survival data
import pandas as pd
import numpy as np

# To remove the scientific notation from numpy arrays
np.set_printoptions(suppress=True)

TitanicSurvivalDataNumeric=pd.read_pickle('TitanicSurvivalDataNumeric.pkl')
TitanicSurvivalDataNumeric.head()

# Separate Target Variable and Predictor Variables
TargetVariable=['Survived']
Predictors=['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
        'Embarked_C', 'Embarked_Q', 'Embarked_S']

X=TitanicSurvivalDataNumeric[Predictors].values
y=TitanicSurvivalDataNumeric[TargetVariable].values


### Sandardization of data ###
### We does not standardize the Target variable for classification
from sklearn.preprocessing import StandardScaler
PredictorScaler=StandardScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X and y
X=PredictorScalerFit.transform(X)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Quick sanity check with the shapes of Training and Testing datasets
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

#Build Artificial Neural Network
#Import the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
```

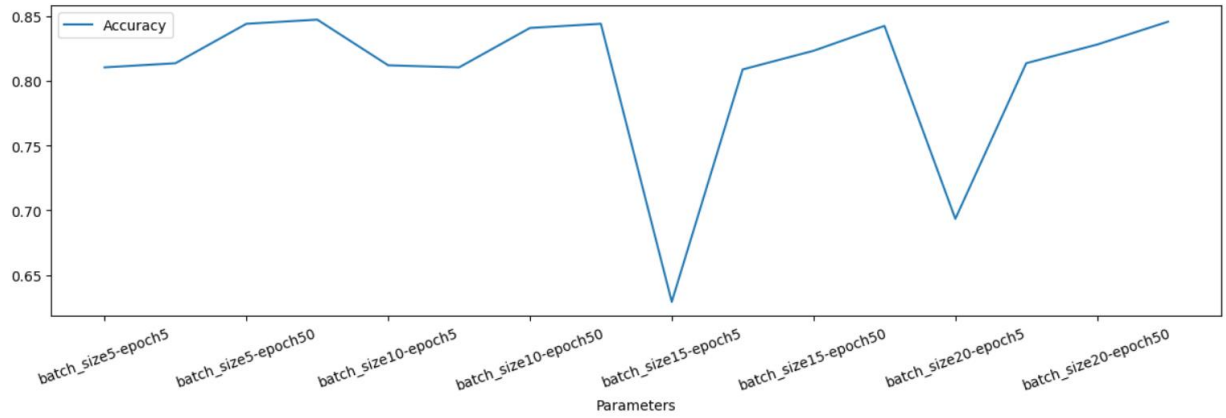
```
classifier = Sequential()
# Defining the Input layer and FIRST hidden layer,both are same!
# relu means Rectifier linear unit function
classifier.add(Dense(units=10, input_dim=9, kernel_initializer='uniform', activation='relu'))

#Defining the SECOND hidden layer, here we have not defined input because it is
# second layer and it will get input as the output of first hidden layer
classifier.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))

# Defining the Output layer
# sigmoid means sigmoid activation function
# for Multiclass classification the activation ='softmax'
# And output_dim will be equal to the number of factor levels
classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))

# Optimizer== the algorithm of SGG to keep updating weights
# loss== the loss function to measure the accuracy
# metrics== the way we will compare the accuracy after each step of SGD
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# fitting the Neural Network on the training data
survivalANN_Model=classifier.fit(X_train,y_train, batch_size=10 , epochs=10, verbose=1)

# fitting the Neural Network on the training data
survivalANN_Model=classifier.fit(X_train,y_train, batch_size=10 , epochs=10, verbose=1)

# Defining a function for finding best hyperparameters
def FunctionFindBestParams(X_train, y_train):

    # Defining the list of hyper parameters to try
    TrialNumber=0
    batch_size_list=[5, 10, 15, 20]
    epoch_list=[5, 10, 50 ,100]

    import pandas as pd
    SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', 'Accuracy'])

    for batch_size_trial in batch_size_list:
        for epochs_trial in epoch_list:
            TrialNumber+=1

            # Creating the classifier ANN model
            classifier = Sequential()
            classifier.add(Dense(units=10, input_dim=9, kernel_initializer='uniform', activation='relu'))
            classifier.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))
            classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
            classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


            survivalANN_Model=classifier.fit(X_train,y_train, batch_size=batch_size_trial , epochs=epochs_trial, verbose=0)
            # Fetching the accuracy of the training
            Accuracy = survivalANN_Model.history['accuracy'][-1]
```

```
            # printing the results of the current iteration
            print(TrialNumber, 'Parameters:','batch_size:', batch_size_trial,'-', 'epochs:',epochs_trial, 'Accuracy:', Accuracy)

            SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber,
                        'batch_size'+str(batch_size_trial)+'-'+'epoch'+str(epochs_trial), Accuracy]],
                                                    columns=['TrialNumber', 'Parameters', 'Accuracy'] ))
    return(SearchResultsData)

# Calling the function
ResultsData=FunctionFindBestParams(X_train, y_train)

# Printing the best parameter
print(ResultsData.sort_values(by='Accuracy', ascending=False).head(1))

# Visualizing the results
%matplotlib inline
ResultsData.plot(x='Parameters', y='Accuracy', figsize=(15,4), kind='line', rot=20)


# Training the model with best hyperparamters
classifier.fit(X_train,y_train, batch_size=5 , epochs=100, verbose=1)

# Predictions on testing data
Predictions=classifier.predict(X_test)

# Scaling the test data back to original scale
Test_Data=PredictorScalerFit.inverse_transform(X_test)

# Generating a data frame for analyzing the test data
TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Survival']=y_test
TestingData['PredictedSurvivalProb']=Predictions

# Defining the probability threshold
def probThreshold(inpProb):
    if inpProb  >  0.5:
        return(1)
    else:
        return(0)

# Generating predictions on the testing data by applying probability threshold
TestingData['PredictedSurvival']=TestingData['PredictedSurvivalProb'].apply(probThreshold)
print(TestingData.head())

from sklearn import metrics
print('\n########## Testing Accuracy Results ##########')
print(metrics.classification_report(TestingData['Survival'], TestingData['PredictedSurvival']))
print(metrics.confusion_matrix(TestingData['Survival'], TestingData['PredictedSurvival']))



# Function to generate Deep ANN model
def make_classification_ann(Optimizer_Trial, Neurons_Trial):
```

```
    from keras.models import Sequential
    from keras.layers import Dense

    # Creating the classifier ANN model
    classifier = Sequential()
    classifier.add(Dense(units=Neurons_Trial, input_dim=9, kernel_initializer='uniform', activation='relu'))
    classifier.add(Dense(units=Neurons_Trial, kernel_initializer='uniform', activation='relu'))
    classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
    classifier.compile(optimizer=Optimizer_Trial, loss='binary_crossentropy', metrics=['accuracy'])

    return classifier


from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier


Parameter_Trials={'batch_size':[10,20,30],
                  'epochs':[10,20],
                'Optimizer_Trial':['adam', 'rmsprop'],
                'Neurons_Trial': [5,10]
               }

# Creating the classifier ANN
classifierModel=KerasClassifier(make_classification_ann, verbose=0)


# Creating the Grid search space
# See different scoring methods by using sklearn.metrics.SCORERS.keys()
grid_search=GridSearchCV(estimator=classifierModel, param_grid=Parameter_Trials, scoring='f1', cv=5)

# Measuring how much time it took to find the best params
import time
StartTime=time.time()

# Running Grid Search for different paramenters
grid_search.fit(X_train,y_train, verbose=1)

EndTime=time.time()
print("################ Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes ##############')

# printing the best parameters
print('\n#### Best hyperparamters ####')
grid_search.best_params_
```

Output:

```
  TrialNumber           Parameters  Accuracy
0           4  batch_size5-epoch100  0.847512
<Axes: xlabel='Parameters'>
```




```
######### Testing Accuracy Results #########
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       157
           1       0.81      0.66      0.73       111

    accuracy                           0.79       268
   macro avg       0.80      0.77      0.78       268
weighted avg       0.80      0.79      0.79       268

[[140  17]
 [ 38  73]]
```

```
Epoch 20/20
21/21 [==============================] - 0s 2ms/step - loss: 0.4404 - accuracy: 0.8074
################ Total Time Taken:  3 Minutes ##############

#### Best hyperparamters ####
{'Neurons_Trial': 10,
 'Optimizer_Trial': 'adam',
 'batch_size': 30,
 'epochs': 20}
```

Experiment 18

Write a program for ANN regression predicting Car Prize.

```
from google.colab import files
uploaded=files.upload()

# Reading the cleaned numeric car prices data
import pandas as pd
import numpy as np

# To remove the scientific notation from numpy arrays
np.set_printoptions(suppress=True)


CarPricesDataNumeric=pd.read_pickle('CarPricesData.pkl')
CarPricesDataNumeric.head()

# Separate Target Variable and Predictor Variables
TargetVariable=['Price']
Predictors=['Age', 'KM', 'Weight', 'HP', 'MetColor', 'CC', 'Doors']

X=CarPricesDataNumeric[Predictors].values
y=CarPricesDataNumeric[TargetVariable].values

### Sandardization of data ###
from sklearn.preprocessing import StandardScaler
PredictorScaler=StandardScaler()
TargetVarScaler=StandardScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)
TargetVarScalerFit=TargetVarScaler.fit(y)

# Generating the standardized values of X and y
X=PredictorScalerFit.transform(X)
y=TargetVarScalerFit.transform(y)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Quick sanity check with the shapes of Training and testing datasets
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
# importing the libraries
from keras.models import Sequential
from keras.layers import Dense

# create ANN model
model = Sequential()

# Defining the Input layer and FIRST hidden layer, both are same!The term kernel_initializer is a fancy term
# for which statistical distribution or function to use for initialising the weights
model.add(Dense(units=5, input_dim=7, kernel_initializer='normal', activation='relu'))

# Defining the Second layer of the model
# after the first layer we don't have to specify input_dim as keras configure it automatically
model.add(Dense(units=5, kernel_initializer='normal', activation='tanh'))

# The output neuron is a single fully connected node
# Since we will be predicting a single number
model.add(Dense(1, kernel_initializer='normal'))

# Compiling the model
#Adam Optimizer is a technique that reduces the time taken to train a model in Deep Learning.
model.compile(loss='mean_squared_error', optimizer='adam')

# Fitting the ANN to the Training set
#verbose = 1, which includes both progress bar and one line per epoch

#verbose = 0, means silent

# one line per epoch i.e. epoch no./total no. of epochs
model.fit(X_train, y_train ,batch_size = 20, epochs = 50, verbose=1)

model.summary()

# Defining a function to find the best parameters for ANN
def FunctionFindBestParams(X_train, y_train, X_test, y_test):

    # Defining the list of hyper parameters to try
    batch_size_list=[5, 10, 15, 20]
    epoch_list  =   [5, 10, 50, 100]

    import pandas as pd
    SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', 'Accuracy'])

    # initializing the trials
    TrialNumber=0
    for batch_size_trial in batch_size_list:
        for epochs_trial in epoch_list:
            TrialNumber+=1
            # create ANN model
            model = Sequential()
            # Defining the first layer of the model
            model.add(Dense(units=5, input_dim=X_train.shape[1], kernel_initializer='normal', activation='relu'))
```

```
        # Defining the Second layer of the model
        model.add(Dense(units=5, kernel_initializer='normal', activation='relu'))

        # The output neuron is a single fully connected node
        # Since we will be predicting a single number
        model.add(Dense(1, kernel_initializer='normal'))

        # Compiling the model
        model.compile(loss='mean_squared_error', optimizer='adam')

        # Fitting the ANN to the Training set
        model.fit(X_train, y_train ,batch_size = batch_size_trial, epochs = epochs_trial, verbose=0)

        MAPE = np.mean(100 * (np.abs(y_test-model.predict(X_test))/y_test))

        # printing the results of the current iteration
        print(TrialNumber, 'Parameters:','batch_size:', batch_size_trial,'-', 'epochs:',epochs_trial, 'Accuracy:', 100-MAPE)

        SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-
'+str(epochs_trial), 100-MAPE]],
                                                    columns=['TrialNumber', 'Parameters', 'Accuracy'] ))
    return(SearchResultsData)


############################################################
# Calling the function
ResultsData=FunctionFindBestParams(X_train, y_train, X_test, y_test)

%matplotlib inline
ResultsData.plot(x='Parameters', y='Accuracy', figsize=(15,4), kind='line')

# Fitting the ANN to the Training set
model.fit(X_train, y_train ,batch_size = 15, epochs = 5, verbose=0)

# Generating Predictions on testing data
Predictions=model.predict(X_test)

# Scaling the predicted Price data back to original price scale
Predictions=TargetVarScalerFit.inverse_transform(Predictions)

# Scaling the y_test Price data back to original price scale
y_test_orig=TargetVarScalerFit.inverse_transform(y_test)

# Scaling the test data back to original scale
Test_Data=PredictorScalerFit.inverse_transform(X_test)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Price']=y_test_orig
TestingData['PredictedPrice']=Predictions
TestingData.head()

# Computing the absolute percent error
APE=100*(abs(TestingData['Price']-TestingData['PredictedPrice'])/TestingData['Price'])
```

```
TestingData['APE']=APE

print('The Accuracy of ANN model is:', 100-np.mean(APE))
TestingData.head()

# Function to generate Deep ANN model
def make_regression_ann(Optimizer_trial):
    from keras.models import Sequential
    from keras.layers import Dense

    model = Sequential()
    model.add(Dense(units=5, input_dim=7, kernel_initializer='normal', activation='relu'))
    model.add(Dense(units=5, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer=Optimizer_trial)
    return model

############################################
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasRegressor

# Listing all the parameters to try
Parameter_Trials={'batch_size':[10,20,30],
                  'epochs':[10,20],
                 'Optimizer_trial':['adam', 'rmsprop']
                }

# Creating the regression ANN model
RegModel=KerasRegressor(make_regression_ann, verbose=0)

############################################
from sklearn.metrics import make_scorer

# Defining a custom function to calculate accuracy
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

########################################
# Creating the Grid search space
# See different scoring methods by using sklearn.metrics.SCORERS.keys()
grid_search=GridSearchCV(estimator=RegModel,
                         param_grid=Parameter_Trials,
                         scoring=custom_Scoring,
                         cv=5)

########################################
# Measuring how much time it took to find the best params
import time
StartTime=time.time()
```

```
# Running Grid Search for different paramenters
grid_search.fit(X,y, verbose=1)

EndTime=time.time()
print("########## Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes')

print('### Printing Best parameters ###')
grid_search.best_params_
```

Output:

```
The Accuracy of ANN model is: 90.89587350466296
```

:

	Age	KM	Weight	HP	MetColor	CC	Doors	Price	PredictedPrice	APE
0	59.0	80430.0	1065.0	110.0	1.0	1600.0	3.0	9950.0	9805.813477	1.449111
1	62.0	64797.0	1075.0	110.0	1.0	1600.0	5.0	7995.0	9977.549805	24.797371
2	59.0	130000.0	1135.0	72.0	1.0	2000.0	4.0	7500.0	8956.398438	19.418646
3	69.0	42800.0	1050.0	110.0	1.0	1600.0	3.0	9950.0	9056.185547	8.983060
4	65.0	47014.0	1015.0	86.0	1.0	1300.0	3.0	8950.0	8812.558594	1.535658

Experiment 19

Write a program Youtube Sentiment Analysis.

Code:

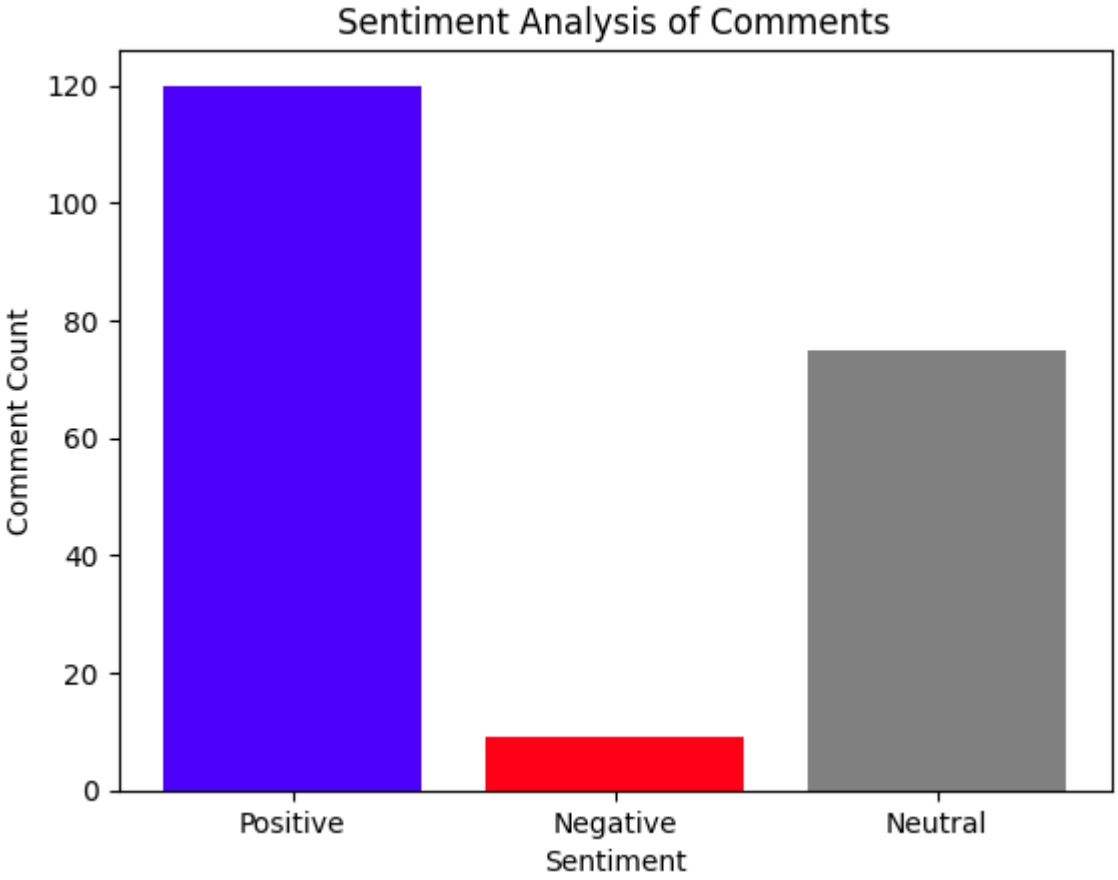
```
pip install google-api-python-client

!pip install emoji
!pip install vaderSentiment

from googleapiclient.discovery import build
# For filtering comments
import re
# For filtering comments with just emojis
import emoji
# Analyze the sentiments of the comment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
# For visualization
import matplotlib.pyplot as plt

API_KEY = 'AIzaSyD8ThZkoMkTrzV488zcMPojU938lfvLaX8'# Put in your API Key

youtube = build('youtube', 'v3', developerKey=API_KEY) # initializing Youtube API

# Taking input from the user and slicing for video id
video_id = input('Enter Youtube Video URL: ')[-11:]
print("video id: " + video_id)

# Getting the channelId of the video uploader
video_response = youtube.videos().list(
    part='snippet',
    id=video_id
).execute()

# Splitting the response for channelID
video_snippet = video_response['items'][0]['snippet']
uploader_channel_id = video_snippet['channelId']
print("channel id: " + uploader_channel_id)

# Fetch comments
print("Fetching Comments...")
comments = []
nextPageToken = None
while len(comments) < 600:
    request = youtube.commentThreads().list(
```

```
        part='snippet',
        videoId=video_id,
        maxResults=100,  # You can fetch up to 100 comments per request
        pageToken=nextPageToken
    )
    response = request.execute()
    for item in response['items']:
        comment = item['snippet']['topLevelComment']['snippet']
        # Check if the comment is not from the video uploader
        if comment['authorChannelId']['value'] != uploader_channel_id:
            comments.append(comment['textDisplay'])
    nextPageToken = response.get('nextPageToken')

    if not nextPageToken:
        break
# Print the 5 comments
comments[:5]

hyperlink_pattern = re.compile(
    r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')

threshold_ratio = 0.65

relevant_comments = []

# Inside your loop that processes comments
for comment_text in comments:

    comment_text = comment_text.lower().strip()

    emojis = emoji.emoji_count(comment_text)

    # Count text characters (excluding spaces)
    text_characters = len(re.sub(r'\s', '', comment_text))

    if (any(char.isalnum() for char in comment_text)) and not hyperlink_pattern.search(comment_text):
        if emojis == 0 or (text_characters / (text_characters + emojis)) > threshold_ratio:
            relevant_comments.append(comment_text)

# Print the relevant comments
relevant_comments[:5]

f = open("ytcomments.txt", 'w', encoding='utf-8')
for idx, comment in enumerate(relevant_comments):
    f.write(str(comment)+"\n")
f.close()
print("Comments stored successfully!")

def sentiment_scores(comment, polarity):

    # Creating a SentimentIntensityAnalyzer object.
```

```
    sentiment_object = SentimentIntensityAnalyzer()

    sentiment_dict = sentiment_object.polarity_scores(comment)
    polarity.append(sentiment_dict['compound'])

    return polarity

polarity = []
positive_comments = []
negative_comments = []
neutral_comments = []

f = open("ytcomments.txt", 'r', encoding='`utf-8')
comments = f.readlines()
f.close()
print("Analysing Comments...")
for index, items in enumerate(comments):
    polarity = sentiment_scores(items, polarity)

    if polarity[-1] > 0.05:
        positive_comments.append(items)
    elif polarity[-1] < -0.05:
        negative_comments.append(items)
    else:
        neutral_comments.append(items)

# Print polarity
polarity[:5]

avg_polarity = sum(polarity)/len(polarity)
print("Average Polarity:", avg_polarity)
if avg_polarity > 0.05:
    print("The Video has got a Positive response")
elif avg_polarity < -0.05:
    print("The Video has got a Negative response")
else:
    print("The Video has got a Neutral response")

print("The comment with most positive sentiment:", comments[polarity.index(max(
    polarity))], "with score", max(polarity), "and length", len(comments[polarity.index(max(polarity))]))
print("The comment with most negative sentiment:", comments[polarity.index(min(
    polarity))], "with score", min(polarity), "and length", len(comments[polarity.index(min(polarity))]))

positive_count = len(positive_comments)
negative_count = len(negative_comments)
neutral_count = len(neutral_comments)

# labels and data for Bar chart
labels = ['Positive', 'Negative', 'Neutral']
comment_counts = [positive_count, negative_count, neutral_count]
```

```
# Creating bar chart
plt.bar(labels, comment_counts, color=['blue', 'red', 'grey'])

# Adding labels and title to the plot
plt.xlabel('Sentiment')
plt.ylabel('Comment Count')
plt.title('Sentiment Analysis of Comments')

# Displaying the chart
plt.show()

# labels and data for Bar chart
labels = ['Positive', 'Negative', 'Neutral']
comment_counts = [positive_count, negative_count, neutral_count]

plt.figure(figsize=(10, 6)) # setting size

# plotting pie chart
plt.pie(comment_counts, labels=labels)

# Displaying Pie Chart
plt.show()
```

Output:


Sentiment Analysis of Comments
Comment Count
120
100
80
60
40
20
0
Positive
Negative
Neutral
Sentiment

Experiment 20

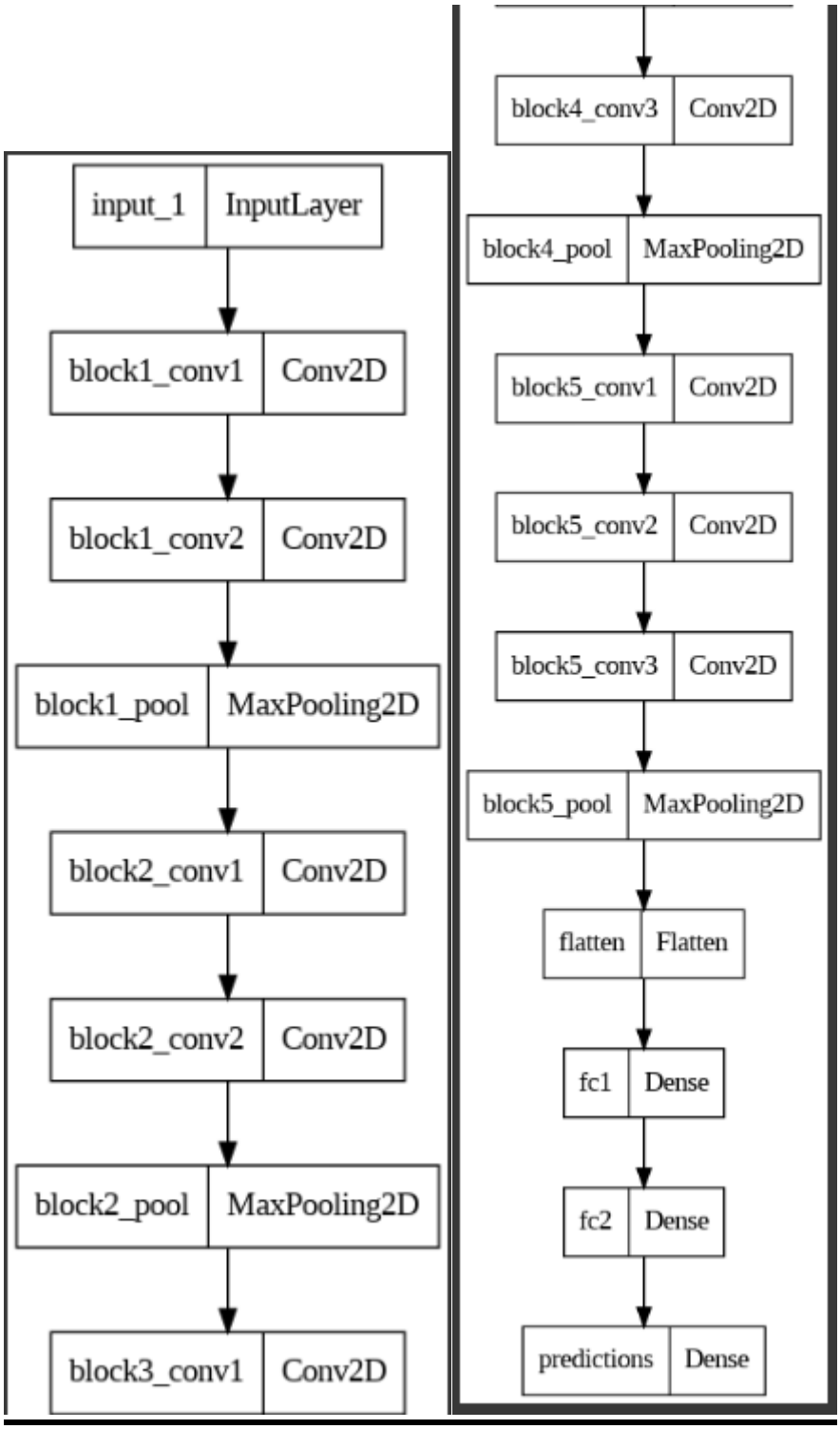
Write a program for Visualizing A CNN Model.

Code:

```
from keras.applications.vgg16 import VGG16
model= VGG16()

import pandas as pd
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import Model
from matplotlib import pyplot
from numpy import expand_dims

#import warnings
#warnings.filterwarnings('ignore')

model.summary()

from keras.utils import plot_model
plot_model(model)

for i in range(len(model.layers)):
  if 'conv' not in model.layers[i].name:
    continue
  filters,biases=model.layers[i].get_weights()
  print('layer number',i,model.layers[i].name,filters.shape)

filter,bias=model.layers[1].get_weights()#2nd layer

#normalize
f_min,f_max=filters.min(),filter.max()
filters=(filters-f_min)/(f_max - f_min)

import matplotlib
n_filters=6
ix=1
fig=pyplot.figure(figsize=(15,10))
for i in range(n_filters):
  f=filters[:,:,:,i]
  for j in range(3):
    pyplot.subplot(n_filters,3,ix)
    pyplot.imshow(f[:,:,j],cmap='gray')
```

```
    ix+=1
pyplot.show()

for i in range(len(model.layers)):
  layer=model.layers[i]
  if 'conv' not in layer.name:
    continue
  print(i,layer.name,layer.output.shape)

model=Model(inputs=model.inputs,outputs=model.layers[1].output)

img=load_img("/content/op.PNG",target_size=(224,224))
#convert
img=img_to_array(img)
img=expand_dims(img,axis=0)
img=preprocess_input(img)

features=model.predict(img)
fig=pyplot.figure(figsize=(20,15))
for i in range(1,features.shape[3]+1):
  pyplot.subplot(8,8,1)
  pyplot.imshow(features[0,:,:,i-1],cmap="gray")
pyplot.show()
```

Output:


input_1 InputLayer
block1_conv1 Conv2D
block1_conv2 Conv2D
block1_pool MaxPooling2D
block2_conv1 Conv2D
block2_conv2 Conv2D
block2_pool MaxPooling2D
block3_conv1 Conv2D
block4_conv3 Conv2D
block4_pool MaxPooling2D
block5_conv1 Conv2D
block5_conv2 Conv2D
block5_conv3 Conv2D
block5_pool MaxPooling2D
flatten Flatten
fc1 Dense
fc2 Dense
predictions Dense

Experiment 21

Write a program for Deep RNN.

Code:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding,SimpleRNN,Dense,LSTM,GRU

(x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=10000)

x_train=pad_sequences(x_train,maxlen=100)
x_test=pad_sequences(x_test,maxlen=100)

#deep Rnn as 2 rnn used
model=Sequential([Embedding(10000,32,input_length=100),
                  SimpleRNN(5,return_sequences=True),
                  SimpleRNN(5),
                  Dense(1,activation='sigmoid')])#BInaryClassificatiom

model.summary()

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

his=model.fit(x_train,y_train,epochs=5,batch_size=32,validation_split=0.2)
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 32)           320000

 simple_rnn_2 (SimpleRNN)    (None, 100, 5)            190

 simple_rnn_3 (SimpleRNN)    (None, 5)                 55

 dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 320251 (1.22 MB)
Trainable params: 320251 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 1/5
625/625 [==============================] - 32s 48ms/step - loss: 0.5688 - accuracy: 0.6909 - val_loss: 0.446
5 - val_accuracy: 0.8052
Epoch 2/5
625/625 [==============================] - 29s 47ms/step - loss: 0.3545 - accuracy: 0.8564 - val_loss: 0.415
0 - val_accuracy: 0.8194
Epoch 3/5
625/625 [==============================] - 29s 46ms/step - loss: 0.2443 - accuracy: 0.9106 - val_loss: 0.443
8 - val_accuracy: 0.8126
Epoch 4/5
625/625 [==============================] - 29s 47ms/step - loss: 0.1538 - accuracy: 0.9511 - val_loss: 0.494
4 - val_accuracy: 0.8090
Epoch 5/5
625/625 [==============================] - 30s 47ms/step - loss: 0.1127 - accuracy: 0.9653 - val_loss: 0.586
3 - val_accuracy: 0.7920
```

Experiment 22

Write a program for Autoencoder Batchsize.

Code:

```
import tensorflow as tf
import numpy as np

def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]

class Autoencoder:
    def __init__(self, input_dim, hidden_dim, epoch=500, batch_size=10, learning_rate=0.001):
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        # Define input placeholder
        x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim])

        # Define variables
        with tf.name_scope('encode'):
            weights = tf.Variable(tf.random_normal([input_dim, hidden_dim], dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            encoded = tf.nn.sigmoid(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim], dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            decoded = tf.matmul(encoded, weights) + biases

        self.x = x
        self.encoded = encoded
        self.decoded = decoded

        # Define cost function and training op
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x, self.decoded))))

        self.all_loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x, self.decoded)), 1))
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)

        # Define a saver op
        self.saver = tf.train.Saver()

    def train(self, data):
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
```

```
        for i in range(self.epoch):
            for j in range(500):
                batch_data = get_batch(data, self.batch_size)
                l, _ = sess.run([self.loss, self.train_op], feed_dict={self.x: batch_data})
            if i % 50 == 0:
                print('epoch {0}: loss = {1}'.format(i, l))
                self.saver.save(sess, './model.ckpt')
        self.saver.save(sess, './model.ckpt')

def test(self, data):
    with tf.Session() as sess:
        self.saver.restore(sess, './model.ckpt')
        hidden, reconstructed = sess.run([self.encoded, self.decoded], feed_dict={self.x: data})
    print('input', data)
    print('compressed', hidden)
    print('reconstructed', reconstructed)
    return reconstructed

def get_params(self):
    with tf.Session() as sess:
        self.saver.restore(sess, './model.ckpt')
        weights, biases = sess.run([self.weights1, self.biases1])
    return weights, biases

def classify(self, data, labels):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        self.saver.restore(sess, './model.ckpt')
        hidden, reconstructed = sess.run([self.encoded, self.decoded], feed_dict={self.x: data})
        reconstructed = reconstructed[0]
        # loss = sess.run(self.all_loss, feed_dict={self.x: data})
        print('data', np.shape(data))
        print('reconstructed', np.shape(reconstructed))
        loss = np.sqrt(np.mean(np.square(data - reconstructed), axis=1))
        print('loss', np.shape(loss))
        horse_indices = np.where(labels == 7)[0]
        not_horse_indices = np.where(labels != 7)[0]
        horse_loss = np.mean(loss[horse_indices])
        not_horse_loss = np.mean(loss[not_horse_indices])
        print('horse', horse_loss)
        print('not horse', not_horse_loss)
        return hidden[7,:]

def decode(self, encoding):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        self.saver.restore(sess, './model.ckpt')
        reconstructed = sess.run(self.decoded, feed_dict={self.encoded: encoding})
    img = np.reshape(reconstructed, (32, 32))
    return img
```

```
from sklearn import datasets

hidden_dim = 1
data = datasets.load_iris().data
input_dim = len(data[0])
ae = Autoencoder(input_dim, hidden_dim)
ae.train(data)
ae.test([[8, 4, 6, 2]])
```

Output:

```
epoch 0: loss = 3.8637373447418213
epoch 50: loss = 0.25829368829727173
epoch 100: loss = 0.3230888843536377
epoch 150: loss = 0.3295430839061737
epoch 200: loss = 0.24636892974376678
epoch 250: loss = 0.22375555336475372
epoch 300: loss = 0.19688692688941956
epoch 350: loss = 0.2520211935043335
epoch 400: loss = 0.29669439792633057
epoch 450: loss = 0.2794385552406311
input [[8, 4, 6, 2]]
compressed [[ 0.72223264]]
reconstructed [[ 6.87640762  2.79334426  6.23228502  2.21386957]]
```

```
array([[ 6.87640762,  2.79334426,  6.23228502,  2.21386957]], dtype=float32)
```

Experiment 23

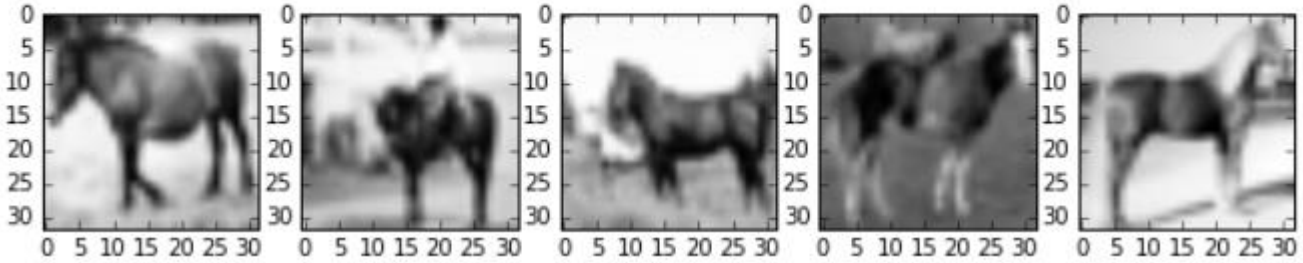
Write a program for Autoencoder with Images.

Code:

```
from matplotlib import pyplot as plt
import pickle
import numpy as np
from autoencoder import Autoencoder

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict

def grayscale(a):
    return a.reshape(a.shape[0], 3, 32, 32).mean(1).reshape(a.shape[0], -1)

names = unpickle('./cifar-10-batches-py/batches.meta')['label_names']
data, labels = [], []
for i in range(1, 6):
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    batch_data = unpickle(filename)
    if len(data) > 0:
        data = np.vstack((data, batch_data['data']))
        labels = np.hstack((labels, batch_data['labels']))
    else:
        data = batch_data['data']
        labels = batch_data['labels']
data = grayscale(data)
x = np.matrix(data)
y = np.array(labels)
Train the autoencoder on images of horses:

horse_indices = np.where(y == 7)[0]
horse_x = x[horse_indices]
print(np.shape(horse_x))  # (5000, 3072)

print('Some examples of horse images we will feed to the autoencoder for training')
plt.rcParams['figure.figsize'] = (10, 10)
num_examples = 5
for i in range(num_examples):
    horse_img = np.reshape(horse_x[i, :], (32, 32))
    plt.subplot(1, num_examples, i+1)
    plt.imshow(horse_img, cmap='Greys_r')
plt.show()
```

Output:

```
(5000, 1024)
Some examples of horse images we will feed to the autoencoder for training
```



```
data (10000, 1024)
reconstructed (1024,)
loss (10000,)
horse 67.4191074286
not horse 65.5469002694
```