EXPERIMENT-1

Title: Learning Rules

Aim: Develop a Python script to execute various learning rules commonly employed in deep learning, including the Hebbian Learning Rule, Perceptron Learning Rule, Delta Learning Rule, Correlation Learning Rule, and OutStar Learning Rule.

Tools: None

Procedure:

1) Initialize the input vectors associated with the target values.

2) Initialize the weights and bias.

3) Set learning rules.

4) Input layer has a unique activation function.

5) Calculate the output.

6) Make adjustments of weights comparing the desired output and target values.

7) Continue the iterations until there is no change of weights.

Program:

```
import numpy as np


# Hebbian Learning Rule

def hebbian_learning_rule(input_pattern, weight_matrix):

    return weight_matrix + np.outer(input_pattern, input_pattern)


# Perceptron Learning Rule
```

```
def perceptron_learning_rule(input_pattern, target, weight_vector, learning_rate):
    prediction = np.dot(weight_vector, input_pattern)
        error = target - prediction
    return weight_vector + learning_rate * error * input_pattern

# Delta Learning Rule
def delta_learning_rule(input_pattern, target, weight_matrix, learning_rate):
    prediction = np.dot(weight_matrix, input_pattern)
        error = target - prediction
    return weight_matrix + learning_rate * np.outer(error, input_pattern)

# Correlation Learning Rule
def correlation_learning_rule(input_pattern, weight_matrix):
    return weight_matrix + np.outer(input_pattern, input_pattern)

# Out Star Learning Rule
def out_star_learning_rule(input_pattern, weight_matrix, learning_rate):
    return weight_matrix + learning_rate * np.outer(input_pattern, input_pattern)

input_size = 3
# Initialize weights with random values
```

```
hebbian_weights = np.random.rand(input_size, input_size)

perceptron_weights = np.random.rand(input_size)

delta_weights = np.random.rand(input_size, input_size)

correlation_weights = np.random.rand(input_size, input_size)

out_star_weights = np.random.rand(input_size, input_size)


print("Hebbian Weights:",hebbian_weights)

print("\nPerceptron  Weights:",perceptron_weights)

print("\nDelta  Weights:",delta_weights)

print("\nCorrelation Weights:",correlation_weights)

print("\nOut Star  Weights:",out_star_weights)
```

output:

```
 # Sample input and target data

input_pattern = np.array([0.2, 0.5, 0.8])

target = 1

 # Apply learning rules

hebbian_weights_updated = hebbian_learning_rule(input_pattern, hebbian_weights)

perceptron_weights_updated = perceptron_learning_rule(input_pattern, target, perceptron_weights, learning_rate=0.1)

delta_weights_updated = delta_learning_rule(input_pattern, target, delta_weights, learning_rate=0.1)

correlation_weights_updated = correlation_learning_rule(input_pattern, correlation_weights)
```

```
out_star_weights_updated = out_star_learning_rule(input_pattern, out_star_weights, learning_rate=0.1)

 # Display updated weights

print("Hebbian Updated Weights:", hebbian_weights_updated)

print("\nPerceptron Updated Weights:", perceptron_weights_updated)

print("\nDelta Updated Weights:", delta_weights_updated)

print("\nCorrelation Updated Weights:",correlation_weights_updated)

print("\nOut Star Updated Weights:",out_star_weights_updated)
```

Output:

EXPERIMENT-2

Title: Activation functions to train Neural Network

Aim: Develop a Python program to implement various activation functions, including the sigmoid, tanh (hyperbolic tangent), ReLU (Rectified Linear Unit), Leaky ReLU, and softmax. The program should include functions to compute the output of each activation function for a given input. Additionally, it should be capable of plotting graphs representing the output of each activation function over a range of input values.

Tools: None

Procedure:

1) Check data that is linearly separable or not.

2) Analyze the activation functions.

3) Set up code for plotting

Program:

```
import numpy as np

import matplotlib.pyplot as plt

def plot_sigmoid():

    x = np.linspace(-10, 10, 100)

    y = 1 / (1 + np.exp(-x))

    plt.plot(x, y)

    plt.xlabel('Input')

    plt.ylabel('Sigmoid Output')

    plt.title('Sigmoid Activation Function')

    plt.grid(True)

    plt.show()
```

```python
def plot_tanh():
    x = np.linspace(-10, 10, 100)
    tanh = np.tanh(x)
    plt.plot(x, tanh)
    plt.title("Hyperbolic Tangent (tanh) Activation Function")
    plt.xlabel("x")
    plt.ylabel("tanh(x)")
    plt.grid(True)
    plt.show()
def plot_relu():
    x = np.linspace(-10, 10, 100)
    relu = np.maximum(0, x)
    plt.plot(x, relu)
    plt.title("ReLU Activation Function")
    plt.xlabel("x")
    plt.ylabel("ReLU(x)")
    plt.grid(True)
    plt.show()
def plot_leaky_relu():
     x = np.linspace(-10, 10, 100)
     def leaky_relu(x, alpha=0.1):
```

```
        return np.where(x >= 0, x, alpha * x)

    # Compute leaky ReLU values for corresponding x

    leaky_relu_values = leaky_relu(x)

    # Plot the leaky ReLU function

    plt.plot(x, leaky_relu_values)

    plt.title("Leaky ReLU Activation Function")

    plt.xlabel("x")

    plt.ylabel("Leaky ReLU(x)")

    plt.grid(True)

    plt.show()

def softmax(z):
    exp_z=np.exp(z)
    class_labels = ["Seal", "Panda", "Duck"]
    soft_ac = [i/ sum(exp_z) for i in exp_z]
    plot_softmax(soft_ac, class_labels)


# Example usage:

    x = np.array([1, 2, 3])

    result = softmax_act(x)

    print(result)

    def plot_softmax(probabilities, class_labels):

        plt.bar(class_labels, probabilities)
```

```
        plt.xlabel("Class")
        plt.ylabel("Probability")
        plt.title("Softmax Output")
        plt.show()


# calling the function
while True:
    print("\nMAIN MENU")
    print("1. Sigmoid")
    print("2. Hyperbolic tangent")
    print("3.Rectified Linear Unit")
    print("4.Leaky ReLU")
    print("5.Softmax")
    print("6.Exit")
    choice = int(input("Enter the Choice:"))
    if choice == 1:
        plot_sigmoid()
    elif choice ==2:
        plot_tanh()
    elif choice ==3:
        plot_relu()
    elif choice ==4:
        plot_leaky_relu()
    elif choice ==5:
```

```
        softmax()
    elif choice ==6:
        break
    else:
        print("Oops! Incorrect Choice.")
```

Output:

EXPERIMENT-3

Title: Perceptron Networks

Aim:Implement a python program for Perceptron Networks by considering the given scenario. A student wants to make a decision about whether to go for a movie or not by looking at 3 parameters using a single neuron. The three inputs are Favorite hero, Exam, and Climate. Each has weights and we have a bias in the perceptron. If the condition is true input is 1 else input is 0, weights for Favorite hero=0.2, Exam=0.4, and Climate=0.2 and bias=-0.5. Output is 1. The decision is to go for a movie.Calculate the Accuracy

Tools: None

Procedure:

1) Initialize the input vector.

2) Train the network weights for the perceptron.

3) Make predictions with the perceptron.

- import the necessary libraries
- Assign the input features to x
- Assign the target features to y
- Initialize the Perceptron with the appropriate number of inputs
- Train the model

- Predict from the test dataset
- Find the accuracy of the model

Program:

```
#import library
import numpy as np

class Perceptron:
    """
    A simple perceptron classifier.
    """
    def __init__(self, weights=None, bias=0):
    self.weights = weights
    self.bias = bias

    def initialize(self, n_features):
    """Set initial w and b as zeros if not provided"""
    if self.weights is None:
    self.weights = np.zeros(n_features)
    if self.bias is None:
    self.bias = 0
    return
```

```
def predict(self, inputs):
"""
Predict the class labels for new input data.
Calculate the step activation function.
"""
activation = np.dot(inputs, self.weights) + self.bias
return 1 if activation > 0 else 0

def train(self, X, y, epochs=100, learning_rate=0.1):
"""Train the perceptron using the input data and target labels."""
# Initialize the weights and bias
self.initialize(X.shape[1])
for epoch in range(epochs):
for inputs, label in zip(X, y):
# Get prediction
y_pred = self.predict(inputs)
# Calculate delta error
error = label - y_pred
# Update weights and bias
self.weights += learning_rate * error * inputs
self.bias += learning_rate * error
return

# Example usage with customized weights
```

```
X_train = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
y_train = np.array([0, 0, 0, 1])
custom_weights = np.array([0.2, 0.4, 0.6])  # Customized weights
custom_bias = -0.5  # Customized bias

p = Perceptron(weights=custom_weights, bias=custom_bias)
p.train(X_train, y_train, epochs=100, learning_rate=0.1)

# Test prediction
test_input = np.array([0, 1, 1])
print("Prediction:", p.predict(test_input))  # Output: 0

# Evaluate accuracy
X_test = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
y_test = np.array([0, 0, 0, 1])

# Predict on test data
pred = np.array([p.predict(x) for x in X_test])
# Calculate accuracy
accuracy = np.mean(pred == y_test) *100
print("Accuracy:", accuracy)
```

Output:

Prediction: 0

Accuracy: 100.0

EXPERIMENT-4

Title:Image processing operations

Aim: Write a program in deep learning to apply image processing operations such as Histogram equalization, Thresholding, Edge detection, Data augmentation, Morphological Operations.
Tools: library opencv-python

Procedure:

1. Load the image as an input
2. Apply image processing operations such as Histogram equalization, Thresholding, Edge detection, Data augmentation, Morphological Operations.
3. Set up code for plotting

Program:

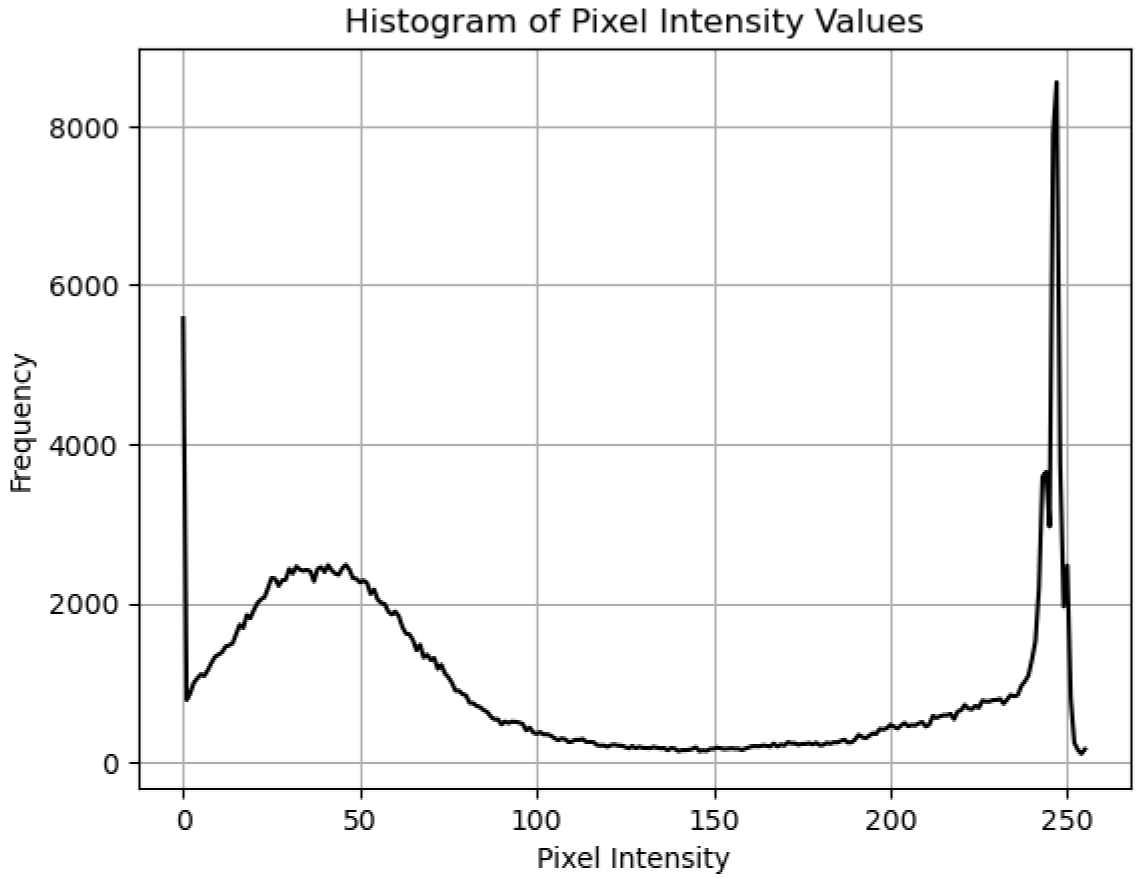
```
#!pip install opencv-python
# Load the libraries
import cv2
import matplotlib.pyplot as plt
import numpy as np

#uploading an image
img = cv2.imread('puppy.jpg')
plt.axis("off")
plt.title("Original Image")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

Output:

Original Image

Compute the histogram and plot

```
histogram = cv2.calcHist([img], [0], None, [256], [0, 256])

plt.plot(histogram, color='black')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.title('Histogram of Pixel Intensity Values')
plt.grid(True)
plt.show()
```




```
# Convert RGB to gray
gray_image = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.imshow(gray_image)
plt.axis("off")
# resizing the image
resized_img = cv2.resize(src =img,
                   dsize=(920, 650),
                   interpolation=cv2.INTER_CUBIC)
plt.imshow(resized_img)
```

#**Displaying the Blurred Image**

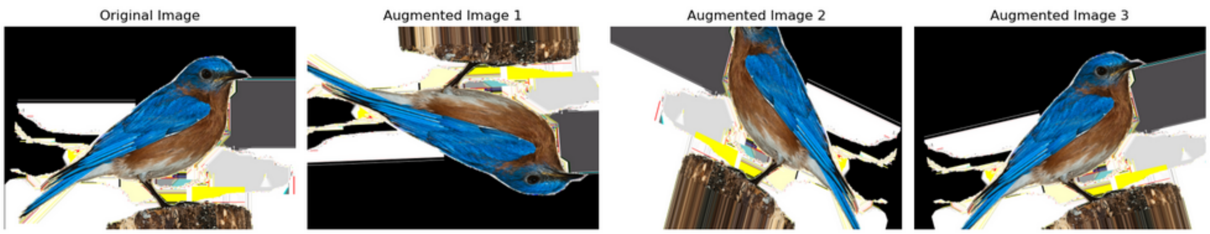
```
gaussian_image = cv2.GaussianBlur(resized_img, (15, 15), 0)
plt.imshow(gaussian_image)
#Canny(image, low_threshold, high_threshold)
```

```
edge = cv2.Canny(resized_img, 100, 200)
plt.imshow(edge)


brightness = cv2.addWeighted(resized_img, 1.2, resized_img, 0, 70)
plt.imshow(brightness)

def sharpen_image(image):
    kernel = np.array([[-1, -1, -1],
                       [-1, 9, -1],
                       [-1, -1, -1]])
    return cv2.filter2D(image, -1, kernel)
sharpened_image = sharpen_image(resized_img)
plt.imshow(resized_img)


original_and_sharpened_image = np.hstack((resized_img, sharpened_image))

plt.figure(figsize = [30, 30])
plt.axis('off')
plt.imshow(original_and_sharpened_image[:,:,::-1])

#data Augmentation

import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
array_to_img, img_to_array, load_img

input_dir = 'data'

# Initialize ImageDataGenerator for augmentation

datagen = ImageDataGenerator(
    rotation_range=20,      # Rotation angle range (degrees)
```

```
    width_shift_range=0.1,  # Fractional shift in the width direction
    height_shift_range=0.1, # Fractional shift in the height direction
    shear_range=0.2,        # Shear intensity (angle in radians)
    zoom_range=0.2,         # Range for random zoom
    horizontal_flip=True,   # Randomly flip inputs horizontally
    vertical_flip=True,     # Randomly flip inputs vertically
    fill_mode='nearest'     # Strategy for filling in newly created pixels
)

# Load an example image to use for augmentation

img = load_img('bird.png')
x = img_to_array(img)
x = np.expand_dims(x, axis=0)

# Generate augmented images

num_images =10
augmented_images = []

# Generate augmented images using the datagen.flow() method

for i, batch in enumerate(datagen.flow(x, batch_size=1)):
    augmented_images.append(array_to_img(batch[0]))
    if i >= num_images - 1:
        break

# Display the original image and augmented images

plt.figure(figsize=(15, 6))
plt.subplot(1, num_images + 1, 1)
plt.imshow(img)
plt.title('Original Image')
plt.axis('off')
```

```
for i in range(num_images):
    plt.subplot(1, num_images + 1, i + 2)
    plt.imshow(augmented_images[i])
    plt.title(f'Augmented Image {i + 1}')
    plt.axis('off')

plt.tight_layout()
plt.show()

# Histogram

import matplotlib.pyplot as plt
from skimage import io, exposure

# Load an image (replace 'input_image.jpg' with your image file path)
input_image = io.imread('puppy.jpg', as_gray=True)

# Apply histogram equalization
/*Histogram Equalization is a technique used in image processing to enhance the
contrast of an image by adjusting the intensity distribution of its pixels*/
equalized_image = exposure.equalize_hist(input_image)

# Display original and equalized images side by side
plt.figure(figsize=(12, 6))

# Plot the original image
plt.subplot(1, 2, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Plot the equalized image
plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Histogram Equalized Image')
```

```
plt.axis('off')

plt.tight_layout()
plt.show()
```




```
 #Morphological Operations (Erosion)
kernel = np.ones((5, 5), np.uint8)
eroded_image = cv2.erode(gray_image, kernel, iterations=1)
plt.subplot(2, 3, 6)
plt.imshow(eroded_image, cmap='gray')
plt.title('Morphological Operations (Erosion)')
plt.axis('off')
```


```
from skimage import filters
import matplotlib.pyplot as plt
# Thresholding (Simple Binary Thresholding)
thresh_value = filters.threshold_otsu(gray_image)
binary_image = gray_image > thresh_value
plt.subplot(2, 3, 3)
plt.imshow(binary_image, cmap='gray')
plt.title('Thresholding')
```

```
plt.axis('off')
```

Thresholding

Experiment No: 5

Title: STYLE TRANSFER FOR AN IMAGE

Aim: Implement image style transfer, transforming a given content image to adopt the artistic style of another image, using a pre-trained model.

Tools: tensor flow and cv2 library

Procedure:

1. Load a STYLE IMAGE and a content Image.
2. Use tensor flow and cv2 library
3. Resizing the style image
4. Apply the Arbitrary Image Stylization" model from TensorFlow Hub
5. Display the output image

Program:

```
#Importing Packages

import tensorflow_hub as hub
import tensorflow as tf
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python.ops.numpy_ops import np_config;
np_config.enable_numpy_behavior()
```

#**Download and Upload the image.**

#I downloaded and uploaded the photos:

- For style: https://imgur.com/9ooB60I
- For content: https://i.imgur.com/F28w3Ac.jpg

#Download the image in google colab

```
!curl https://imgur.com/9ooB60I.jpeg -o style.jpeg

!curl https://i.imgur.com/F28w3Ac.jpg -o content.jpg
```

 # lOAD THE IMAGE

```
def load_img(path):

    img = cv2.imread(path)

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    img = img/255.

    return img

content_image = load_img('style.jpeg')

style_1 = load_img('content.jpeg')

#content_image = load_img('content.jpeg')

#style_1 = load_img('style.jpeg')

model =
hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
```

Apply the style

```
def apply_style(content_image, style):

  content_image = content_image.reshape(1, content_image.shape[0],
content_image.shape[1], content_image.shape[2]).astype('float32')

  content_image = tf.convert_to_tensor(content_image)
```

```
  style = cv2.resize(style, (256,256))

  style = style.reshape(1, style.shape[0], style.shape[1], style.shape[2]).astype('float32')

  outputs = model(tf.constant(content_image), tf.constant(style))

  stylized_image = outputs[0]

  return stylized_image

#display the image

img = apply_style(content_image, style_1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(img[0])

plt.show()
```

Output:

EXPERIMENT-6

Aim: Implement in python SVM/Softmax classifier for CIFAR-10 dataset.

Tools:Tensorflow, CIFAR-10

Procedure:

1. Load the CIFAR-10
2. Preprocess the data
    a. Normalize the pixel values to be between 0 and 1.
    b. Convert the class labels to one-hot encoded vectors.
3. Train the data with SVM classifier
4. Train the data with Softmax classifier using KNN
5. Make predictions with the SVM/Softmax classifier.

Program:

#import library

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.datasets import cifar10
from sklearn.preprocessing import OneHotEncoder
```

classify test and train data
#load the cifar data set X-image, y-label for train and test

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
# define the class name
```
for easier interpretation of predictions.

```
cifar_10_classes = [
    "Airplane",
    "Automobile",
    "Bird",
    "Cat",
    "Deer",
    "Dog",
    "Frog",
    "Horse",
    "Ship",
    "Truck"
]
```

```
X_train.shape
```

```
import matplotlib.pyplot as plt

plt.imshow(x_train[0])
plt.title(cifar_10_classes[y_train[0][0]])
plt.axis("off")
```

#Normalizing the pixel values to be between 0 and 1

```
X_train = x_train / 255.0
X_test = x_test / 255.0
```

#Converting the class labels to one-hot encoded vectors.Converting the class labels to one-hot encoded vectors.

```
one_hot_encoder = OneHotEncoder()
y_train = one_hot_encoder.fit_transform(y_train).toarray()
y_test = one_hot_encoder.transform(y_test).toarray()
```

```
#build softmax layer
```

- Creating a simple neural network model with a single layer.
- `Flatten` layer converts the 32x32x3 images into a flat vector.
- `Dense` layer with `softmax` activation outputs probabilities for each of the 10 classes.

```
softmax_model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
softmax_model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
```

#Training the model for 20 epochs with a batch size of 64, using a validation split for evaluation.

```
softmax_model.fit(X_train, y_train, epochs=20, batch_size=64,
validation_data=(X_test, y_test))
```

#Making a prediction using the trained model

```
new_image = x_test[10]
plt.imshow(new_image)
plt.axis("off")

img = np.expand_dims(new_image, axis=0)

img.shape

pred = softmax_model.predict(img)



pred

prediction = np.argmax(pred)
Cifar_10_classes[prediction]
```

Output:

EXPERIMENT-7

Title: MULTI-LAYER NEURAL NETWORKS

Aim: Develop a convolutional neural network (CNN) model to classify handwritten digits using the MNIST dataset. The goal is to train a model that accurately identifies digits (0-9) from images.

Tools: None

Procedure:

1. Prepare the Data
2. Define the Model
3. Train the Model
4. Evaluate the model
5. Make predictions

Program:

```
#importing libraries

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

import tensorflow.keras as keras

import numpy as np


#load the dataset and  divide into train and test print the shape and size

dataset = keras.datasets.mnist

class_names = ['Zero','one','two','three','Four','Five','Six','seven','Eight','nine']

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

X_train = x_train.reshape((x_train.shape[0], x_train.shape[1], x_train.shape[2], 1))

X_test = x_test.reshape((x_test.shape[0],x_test.shape[1],x_test.shape[2],1))
```

```
print(X_train.shape)
print(X_test.shape)

# plot five data with its class name
plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train[i])
    plt.title(class_names[y_train[i]])
    plt.axis("off")


#convert into grayscale
X_train=X_train/255
X_test=X_test/255


#Model
model = keras.models.Sequential([
    keras.layers.Conv2D(64,(3,3),input_shape=(28,28,1),activation="relu"),
    keras.layers.MaxPool2D(pool_size=(2,2),strides=1),
    keras.layers.Conv2D(64,(3,3),input_shape=(28,28,1),activation="relu"),
    keras.layers.MaxPool2D(pool_size=(2,2),strides=1),
    keras.layers.Flatten(),
    keras.layers.Dense(64,activation="relu"),
```

```
    keras.layers.Dense(10,activation="softmax")
])

model.compile(optimizer="adam",loss=tf.keras.losses.SparseCategoricalCrossentr
opy(from_logits=True),metrics=["accuracy"])

model.fit(x_train,y_train,epochs=5,callbacks=keras.callbacks.EarlyStopping(patien
ce=2))

#evaluting the model
model.evaluate(x_test,y_test)



#Prediction
sample_img = X_test[0]
sample_img.shape
plt.imshow(sample_img)

img = np.expand_dims(sample_img,axis=0)
img.shape

pred = model.predict(img)

pred
```

```
print(f"Predicted:                    {class_names[np.argmax(pred)]}\nActual:
{class_names[y_test[0]]}")


model.summary()
```

EXPERIMENT-8

Title: **Dropout Regularization In Deep Neural Network**

Aim: Design and implement a deep learning model to classify underwater sonar signals into two categories (Rocks 'R' or Mines 'M') using the `sonar_dataset.csv`. Evaluate the performance of the model on unseen test data and demonstrate the impact of incorporating dropout layers to improve generalization.

Tools: Tensorflow

Procedure:

1. Load the dataset `sonar_dataset.csv`
2. Train the model with configuration drop out
3. Evaluate models on test data

Program

```
#import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv("sonar_dataset.csv", header=None)
df.sample(5)
```

```
df.shape

# check for nan values
df.isna().sum()

df.columns

df[60].value_counts() # label is not skewed
X = df.drop(60, axis=1)
y = df[60]
y.head()


y = pd.get_dummies(y, drop_first=True)
y.sample(5) # R --> 1 and M --> 0

y.value_counts()

X.head()


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=1)

X_train.head()
```

Using Deep Learning Model

Model without Dropout Layer

```
import tensorflow as tf
```

```
from tensorflow import keras


model = keras.Sequential([
    keras.layers.Dense(60, input_dim=60, activation='relu'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

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

model.fit(X_train, y_train, epochs=100, batch_size=8)

model.evaluate(X_test, y_test)

y_pred = model.predict(X_test).reshape(-1)
print(y_pred[:10])

# round the values to nearest integer ie 0 or 1
y_pred = np.round(y_pred)
print(y_pred[:10])

y_test[:10]


from sklearn.metrics import confusion_matrix , classification_report

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

Model with Dropout Layer

```
modeld = keras.Sequential([
```

```python
    keras.layers.Dense(60, input_dim=60, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')
])

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

modeld.fit(X_train, y_train, epochs=100, batch_size=8)

modeld.evaluate(X_test, y_test)
```

Training Accuracy is still good but Test Accuracy Improved

```python
y_pred = modeld.predict(X_test).reshape(-1)
print(y_pred[:10])

# round the values to nearest integer ie 0 or 1
y_pred = np.round(y_pred)
print(y_pred[:10])


from sklearn.metrics import confusion_matrix , classification_report

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

Output:

You can see that by using dropout layer test accuracy increased from 0.77 to 0.81

EXPERIMENT-9

Title: Image segmentation using mask RCNN

Aim: To implement an object detection and segmentation model using Mask R-CNN on a given image.

Tools:Tensorflow

Procedure:

1. **Set Up the Environment:**
    - Clone the Mask R-CNN repository and set up the necessary libraries and modules.
    - Define the root directory for the project and add the Mask R-CNN module to the system path.
    - Specify the directory to save logs and the trained model.
2. **Load and Configure the Model:**
    - Load the Mask R-CNN model in inference mode.
    - Download the pre-trained weights if they do not exist in the specified path.
    - Configure the model for inference by setting parameters like the number of GPUs and images per GPU.
3. **Define Class Names:**
    - Use the class names from the COCO dataset for easier interpretation of the model's predictions.
4. **Load and Visualize the Image:**
    - Upload and read the input image using skimage.
    - Display the original image for reference.
5. **Perform Object Detection and Segmentation:**
    - Use the Mask R-CNN model to detect objects in the image.
    - Display the results, including bounding boxes, class labels, segmentation masks, and confidence scores for each detected object.

Program:

```
!git clone https://github.com/akTwelve/Mask_RCNN.git
```

#import libraries

```python
import os
import sys
import skimage.io
import matplotlib.pyplot as plt
import cv2
import time
import numpy as np
import tensorflow as tf

# Root directory of the project
ROOT_DIR = "Mask_RCNN"

# Import maskrcnn (mrcnn folder) as module
sys.path.append(ROOT_DIR)
```

Make sure to upload the Image and you need to be inside MASK_RCNN folder

to run from mrcnn. So keep in kind about the path

```python
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize

# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# upload image path
IMAGE_PATH = "/content/Mask_RCNN/images/1045023827_4ec3e8ba5c_z.jpg"
```

#Download the COCO dataset

```
sys.path.append(os.path.join(ROOT_DIR, "samples/coco/"))

import coco


# Weights oath of Mask RCNN

COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")


if not os.path.exists(COCO_MODEL_PATH):

    utils.download_trained_weights(COCO_MODEL_PATH)


# Loading the model configuration

class InferenceConfig(coco.CocoConfig):

    GPU_COUNT = 1

    IMAGES_PER_GPU = 1


config = InferenceConfig()

config.display()
```

```
model     =     modellib.MaskRCNN(mode="inference",     model_dir=MODEL_DIR,
config=config)


tf.keras.Model.load_weights(model.keras_model,            COCO_MODEL_PATH,
by_name=True)

class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',

               'bus', 'train', 'truck', 'boat', 'traffic light',

                   'fire hydrant', 'stop sign', 'parking meter', 'bench',
'bird',

               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
```

```
                    'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag',
'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
                    'surfboard', 'tennis racket', 'bottle', 'wine glass',
'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
                    'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog',
'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
                      'dining table', 'toilet', 'tv', 'laptop', 'mouse',
'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
                         'sink', 'refrigerator', 'book', 'clock', 'vase',
'scissors',
               'teddy bear', 'hair drier', 'toothbrush']
```

```
image = skimage.io.imread(IMAGE_PATH)

plt.imshow(image)

plt.title('Original')

plt.axis('off')

plt.show()
```

Original

```
import numpy as np
np.bool = np.bool_
results = model.detect([image], verbose=1)

r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            class_names, r['scores'])
```

```
image_metas              shape: (1, 93)             min:    0.00000  max: 1024.00000  float64
anchors                  shape: (1, 261888, 4)      min:   -0.35390  max:    1.29134  float32
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training_v1.py:2359: UserWarning: `Model.state_updates` will be removed in a future version. This property shoul
  updates=self.state_updates,
```

EXPERIMENT-10

Title:Study the effect of batch normalization and dropout in neural network classifier

Aim: The aim of the study of the effect of batch normalization and dropout in neural network classifiers is to build, train, and evaluate a deep neural network model for the classification of handwritten digits using the MNIST dataset. The MNIST dataset is a widely recognized benchmark for image classification algorithms, consisting of 60,000 training images and 10,000 test images of handwritten digits (0-9).

Tools: None

Procedure:

1. **Load and Preprocess the Data:**
    - Load the MNIST dataset.
    - Normalize the pixel values of the images to be between 0 and 1.
2. Build the Neural Network Model

3. Compile the Model
4. Train the Model
5. Evaluate the Model

Program:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, BatchNormalization, 
Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train / 255.0
X_test = X_test / 255.0
model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        BatchNormalization(),  # Adding Batch Normalization layer
        Dropout(0.2),          # Adding Dropout layer with dropout rate of 
0.2
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])
model.compile(optimizer="adam",
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32, 
validation_data=(X_test, y_test))
train_loss, train_acc = model.evaluate(X_train, y_train)
print("Train Loss:", train_loss)
print("Train accuracy:", train_acc)
```

Output:

```
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0298 - accuracy: 0.9905
Train Loss: 0.029764531180262566
Train accuracy: 0.9904999732971191
```

```python
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0662 - accuracy: 0.9793
Test Loss: 0.06620358675718307
Test accuracy: 0.9793000221252441
```

EXPERIMENT-11

Aim: Chatbots using bidirectional LSTMs Tools

Tools: LSTMs

Procedure:

1. Define a simple dataset of conversation pairs, create a vocabulary from the dataset and convert conversations into numerical sequences.
2. Pad sequences have equal lengths.
3. Define a bidirectional LSTM model with an embedding layer and a dense output layer.
4. Train the model on the conversation data.
5. Define a function to generate responses based on user inputs.

```
Program:
import os
from zipfile import ZipFile
from IPython.display import FileLink

# Assuming you manually uploaded the kaggle.json file to your Jupyter
environment

# Check if the kaggle.json file is present
if 'kaggle.json' not in os.listdir():
    print("Please upload your Kaggle API key (kaggle.json).")

# Install the Kaggle package
!pip install -q kaggle

# Create Kaggle directory and move the API key file
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download the dataset
!kaggle datasets download -d kausr25/chatterbotenglish

# Unzip the downloaded file
with ZipFile('chatterbotenglish.zip', 'r') as zip_ref:
    zip_ref.extractall()

# Check the contents of the directory
dir_path = '.'
data = os.listdir(dir_path)
print(data)
import yaml
```

```
import os

dir_path = '/content'
data = os.listdir(dir_path + os.sep)
data

files_list = []
for i in data:
  if i.endswith(('.yml', '.yaml')):
    files_list.append(i)
files_list


questions, answers = [], []

for filepath in files_list:
    file_ = open(dir_path + os.sep + filepath , 'rb')
    docs = yaml.safe_load(file_)
    conversations = docs['conversations']
    for con in conversations:
        if len(con) > 2 :
            questions.append(con[0])
            replies = con[1 :]
            ans = ''
            for rep in replies:
                ans += ' ' + rep
            answers.append(ans)
        elif len(con)> 1:
            questions.append(con[0])
            answers.append(con[1])
questions

import numpy as np
from tensorflow.keras import layers , activations , models , preprocessing,
utils

answers_with_tags = []
for i in range(len(answers)):
    if type(answers[i]) == str:
        answers_with_tags.append(answers[i])
    else:
        questions.pop(i)

answers = []
for i in range(len(answers_with_tags)) :
    answers.append('<START> ' + answers_with_tags[i] + ' <END>')

tokenizer = preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(questions + answers)
```

```
VOCAB_SIZE = len(tokenizer.word_index)+1

from gensim.models import Word2Vec
import re

vocab = []
for word in tokenizer.word_index:
    vocab.append(word)

def tokenize(sentences):
    tokens_list = []
    vocabulary = []
    for sentence in sentences:
        sentence = sentence.lower()
        sentence = re.sub('[^a-zA-Z]', ' ', sentence)
        tokens = sentence.split()
        vocabulary += tokens
        tokens_list.append(tokens)
    return tokens_list , vocabulary
tokenized_questions = tokenizer.texts_to_sequences(questions)
maxlen_questions = max([len(x) for x in tokenized_questions])
padded_questions = preprocessing.sequence.pad_sequences(tokenized_questions ,
maxlen=maxlen_questions , padding='post')
encoder_input_data = np.array(padded_questions)
encoder_input_data = np.array(padded_questions)

tokenized_questions = tokenizer.texts_to_sequences(questions)
maxlen_questions = max([len(x) for x in tokenized_questions])
padded_questions = preprocessing.sequence.pad_sequences(tokenized_questions ,
maxlen=maxlen_questions , padding='post')
encoder_input_data = np.array(padded_questions)


tokenized_answers = tokenizer.texts_to_sequences(answers)
for i in range(len(tokenized_answers)) :
    tokenized_answers[i] = tokenized_answers[i][1:]
padded_answers = preprocessing.sequence.pad_sequences(tokenized_answers ,
maxlen=maxlen_answers , padding='post')
onehot_answers = utils.to_categorical(padded_answers , VOCAB_SIZE)
decoder_output_data = np.array(onehot_answers)


import tensorflow as tf

encoder_inputs = tf.keras.layers.Input(shape=(maxlen_questions ,))
encoder_embedding = tf.keras.layers.Embedding(VOCAB_SIZE, 200 , mask_zero=True)
(encoder_inputs)
encoder_outputs , state_h , state_c = tf.keras.layers.LSTM(200 ,
return_state=True)(encoder_embedding)
```

```
encoder_states = [ state_h , state_c ]

decoder_inputs = tf.keras.layers.Input(shape=(maxlen_answers , ))
decoder_embedding = tf.keras.layers.Embedding(VOCAB_SIZE, 200 , mask_zero=True)
(decoder_inputs)
decoder_lstm = tf.keras.layers.LSTM(200 , return_state=True ,
return_sequences=True)
decoder_outputs , _ , _ = decoder_lstm (decoder_embedding ,
initial_state=encoder_states)
decoder_dense = tf.keras.layers.Dense(VOCAB_SIZE ,
activation=tf.keras.activations.softmax)
output = decoder_dense (decoder_outputs)

model = tf.keras.models.Model([encoder_inputs, decoder_inputs], output)

model.compile(optimizer=tf.keras.optimizers.RMSprop(),
loss='categorical_crossentropy')

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 22)]                 0         []

 input_2 (InputLayer)        [(None, 74)]                 0         []

 embedding (Embedding)       (None, 22, 200)              378800    ['input_1[0][0]']

 embedding_1 (Embedding)     (None, 74, 200)              378800    ['input_2[0][0]']

 lstm (LSTM)                 [(None, 200),                320800    ['embedding[0][0]']
                              (None, 200),
                              (None, 200)]

 lstm_1 (LSTM)               [(None, 74, 200),            320800    ['embedding_1[0][0]',
                              (None, 200),                           'lstm[0][1]',
                              (None, 200)]                           'lstm[0][2]']

 dense (Dense)               (None, 74, 1894)             380694    ['lstm_1[0][0]']

==================================================================================================
Total params: 1779894 (6.79 MB)
Trainable params: 1779894 (6.79 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
model.fit([encoder_input_data , decoder_input_data], decoder_output_data,
batch_size=32, epochs=100)
```

```
model.fit([encoder_input_data , decoder_input_data], decoder_output_data, batch_size=32, epochs=10
18/18 [==============================] - 7s 367ms/step - loss: 4.0626
Epoch 7/100
18/18 [==============================] - 8s 464ms/step - loss: 4.0389
Epoch 8/100
18/18 [==============================] - 7s 375ms/step - loss: 4.0188
Epoch 9/100
18/18 [==============================] - 8s 440ms/step - loss: 3.9917
Epoch 10/100
18/18 [==============================] - 8s 429ms/step - loss: 3.9757
Epoch 11/100
18/18 [==============================] - 7s 386ms/step - loss: 3.9490
Epoch 12/100
18/18 [==============================] - 8s 462ms/step - loss: 3.9360
Epoch 13/100
18/18 [==============================] - 7s 379ms/step - loss: 3.9097
Epoch 14/100
18/18 [==============================] - 8s 454ms/step - loss: 3.8797
Epoch 15/100
18/18 [==============================] - 7s 376ms/step - loss: 3.8593
Epoch 16/100
```

```
encoder_model = tf.keras.models.Model(encoder_inputs, encoder_states)
decoder_state_input_h = tf.keras.layers.Input(shape=(200 ,))
decoder_state_input_c = tf.keras.layers.Input(shape=(200 ,))

decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

decoder_outputs, state_h, state_c = decoder_lstm(
    decoder_embedding , initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = tf.keras.models.Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)
def preprocess_input(input_sentence):
    tokens = input_sentence.lower().split()
    tokens_list = []
    for word in tokens:
        tokens_list.append(tokenizer.word_index[word])
    return preprocessing.sequence.pad_sequences([tokens_list] ,
maxlen=maxlen_questions , padding='post')

tests = ['Hello', 'Are you a bot', 'What is your name', 'That is a very long
name', 'see you later']

for i in range(5):
    states_values = encoder_model.predict(preprocess_input(tests[i]))
    empty_target_seq = np.zeros((1 , 1))
    empty_target_seq[0, 0] = tokenizer.word_index['start']
    stop_condition = False
    decoded_translation = ''
```

```
    while not stop_condition :
        dec_outputs , h , c = decoder_model.predict([empty_target_seq] +
states_values)
        sampled_word_index = np.argmax(dec_outputs[0, -1, :])
        sampled_word = None

        for word , index in tokenizer.word_index.items() :
            if sampled_word_index == index :
                decoded_translation += f' {word}'
                sampled_word = word

        if sampled_word == 'end' or len(decoded_translation.split()) >
maxlen_answers:
            stop_condition = True

        empty_target_seq = np.zeros((1 , 1))
        empty_target_seq[0 , 0] = sampled_word_index
        states_values = [h , c]
    print(f'Human: {tests[i]}')
    print()
    decoded_translation = decoded_translation.split(' end')[0]
    print(f'Bot: {decoded_translation}')
    print('-'*25)
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
Human: Hello

Bot:  i am not really really stock
-------------------------
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
Human: Are you a bot
```

EXPERIMENT-12

Aim:

EXPERIMENT-12

Aim: Object detection with single-stage and two-stage detectors (Yolo)

Tools: YOLO Model

Procedure:

1. Load the Model
2. Pre-process the Image
3. Run Object Detection
4. Post-processing
5. Visualize Results

EXPERIMENT-13

Aim: Implement image captioning with vanilla RNN using seq2seq model

Tools: None

Procedure:

1. Use a simplified version of the model with a single Dense layer as the encoder and a single LSTM layer as the decoder.
2. The image features are extracted separately
3. Use a <start> token to initiate the decoding process and a <end> token to signal the end of the caption.
4. The model learns to generate captions.
5. Generate captions for new images.

EXPERIMENT-14

Aim: To Learn and implement the DCGAN model to simulate realistic images, with IanGoodfellow, the inventor of GANS (generative adversarial networks)

Tools: None

Procedure:

Step 1: Select a number of real images from the training set.

Step 2: Generate a number of fake images. This is done by sampling random noise vectors and creating images from them using the generator.

Step 3: Train the discriminator for one or more epochs using both fake and real images. This will update only the discriminator's weights by labelling all the real images as 1 and the fake images as 0.

Step 4: Generate another number of fake images.

Step 5: Train the full GAN model for one or more epochs using only fake images. This will update only the generator's weights by labelling all fake images as 1.

Program: