Department of CSE (AIML & IOT)

LIST OF EXPERIMENTS

1. Write a program to perform different 2D and 3D transformation techniques on a given image using Python
2. Write a program to simulate different Linear and Non-Linear filters on a given image using Python
3. Write a program to implement Canny edge detection algorithm using Python
4. Write a program to extract interest points of an image by SIFT Detector using Python
5. Write a program to implement Image stiching panorama using Python
6. Write a program to implement Bag-of-words for a given image using Python.
7. Write a program to implement Artificial Neural network to classify MNIST dataset
8. Classification CIFAR-10 Dataset using AlexNet using PyTorch
9. Classification CIFAR-10 Dataset using ZFNet using PyTorch
10. Classification CIFAR-10 Dataset using VGGNet using PyTorch
11. Object detection for PASCAL VOC dataset using YOLO Object Detection using PyTorch
12. Image generation of CIFAR10 datasets using DCGAN using PyTorch

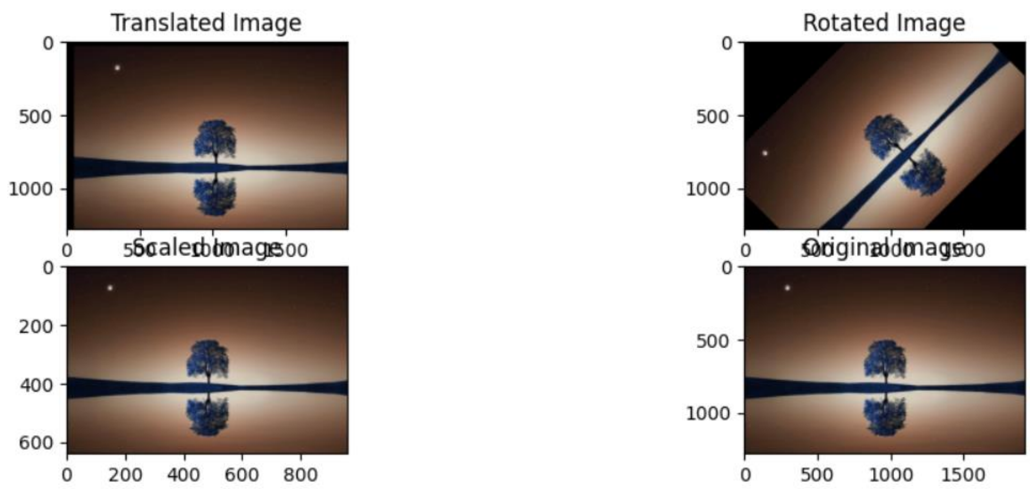
Experiment Number: 1

Title of the Experiment: Write a program to perform different 2D and 3D transformation techniques on a given image using Python

Program:

```
#2D
import cv2
import numpy as np
from matplotlib import pyplot as plt
import io
from PIL import Image

# Load the image
image = cv2.imread('/content/drive/MyDrive/DLCV Internal/Internal6601-1.jpg')

# 2D translation
tx, ty = 50, 30  # Translation in x and y
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated_image = cv2.warpAffine(image, translation_matrix, (image.shape[1], image.shape[0]))

# 2D rotation
angle = 45  # Rotation angle in degrees
rotation_matrix = cv2.getRotationMatrix2D((image.shape[1] / 2, image.shape[0] / 2), angle, 1)
rotated_image = cv2.warpAffine(image, rotation_matrix, (image.shape[1], image.shape[0]))

# 2D scaling
scale_factor = 0.5
scaled_image = cv2.resize(image, None, fx=scale_factor, fy=scale_factor)

# Display the transformed images
plt.figure(figsize=(12, 4))
plt.subplot(221), plt.imshow(translated_image), plt.title('Translated Image')
plt.subplot(222), plt.imshow(rotated_image), plt.title('Rotated Image')
plt.subplot(223), plt.imshow(scaled_image), plt.title('Scaled Image')
plt.subplot(224),plt.imshow(image),plt.title('Original Image')
plt.show()
```

output:

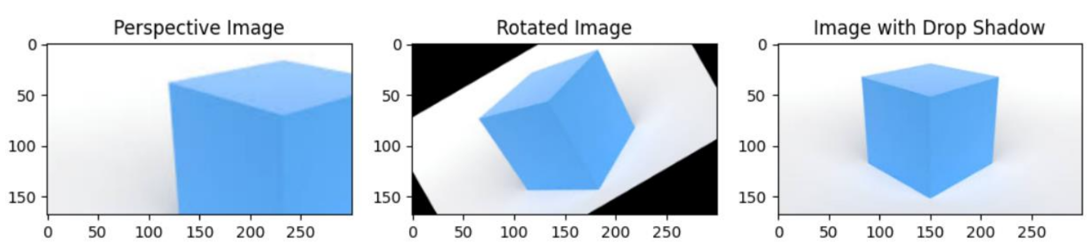


#3D

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import io
from PIL import Image

# Load the image
image = cv2.imread('/content/drive/MyDrive/DLCV Internal/Internal6601-2.jpg')

pts1 = np.float32([[10, 10], [100, 10], [10, 100], [100, 100]])
pts2 = np.float32([[0, 0], [150, 0], [0, 150], [150, 150]])

# Calculate the perspective transformation matrix
perspective_matrix = cv2.getPerspectiveTransform(pts1, pts2)

# Apply the perspective transformation
perspective_image = cv2.warpPerspective(image, perspective_matrix, (image.shape[1],
image.shape[0]))
```

```
# Simulate 3D rotation
angle = 30  # Rotation angle in degrees
rotation_matrix = cv2.getRotationMatrix2D((image.shape[1] / 2, image.shape[0] / 2), angle, 1)
rotated_image = cv2.warpAffine(image, rotation_matrix, (image.shape[1], image.shape[0]))

# Create a drop shadow effect
shadow = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
shadow = cv2.GaussianBlur(shadow, (0, 0), 10)
shadow = cv2.addWeighted(shadow, 2.0, shadow, 0, 0)
image_with_shadow = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
image_with_shadow[:, :, 3] = shadow

# Display the transformed images
plt.figure(figsize=(12, 4))
plt.subplot(131), plt.imshow(cv2.cvtColor(perspective_image, cv2.COLOR_BGR2RGB)),
plt.title('Perspective Image')
plt.subplot(132), plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB)), plt.title('Rotated
Image')
plt.subplot(133), plt.imshow(image_with_shadow), plt.title('Image with Drop Shadow')
plt.show()
```

output:



Department of CSE-AIML & IoT

Experiment Number: 2

Title of the Experiment: Write a program to simulate different Linear and Non-Linear filters on a given image using Python

Program:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

def apply_linear_filter(image, kernel):
  filtered_image = cv2.filter2D(image, -1, kernel)
  return filtered_image

def apply_nonlinear_filter(image, filter_type):
  if filter_type == "median":
    filtered_image = cv2.medianBlur(image, 5)
  elif filter_type == "gaussian":
    filtered_image = cv2.GaussianBlur(image, (5, 5), 0)
  elif filter_type == "bilateral":
    filtered_image = cv2.bilateralFilter(image,15,75,75)
  elif filter_type == "min":
    filtered_image = cv2.erode(image, cv2.getStructuringElement(cv2.MORPH_RECT, (5,5)))
  elif filter_type == "max":
    filtered_image = cv2.dilate(image, cv2.getStructuringElement(cv2.MORPH_RECT, (5,5)))
  else:
    raise ValueError("Invalid non-linear filter type.")
  return filtered_image

image_path = "/content/drive/MyDrive/image1.jpg"
image = cv2.imread(image_path)

fig, axs = plt.subplots(1,3,figsize=(10,5))
```

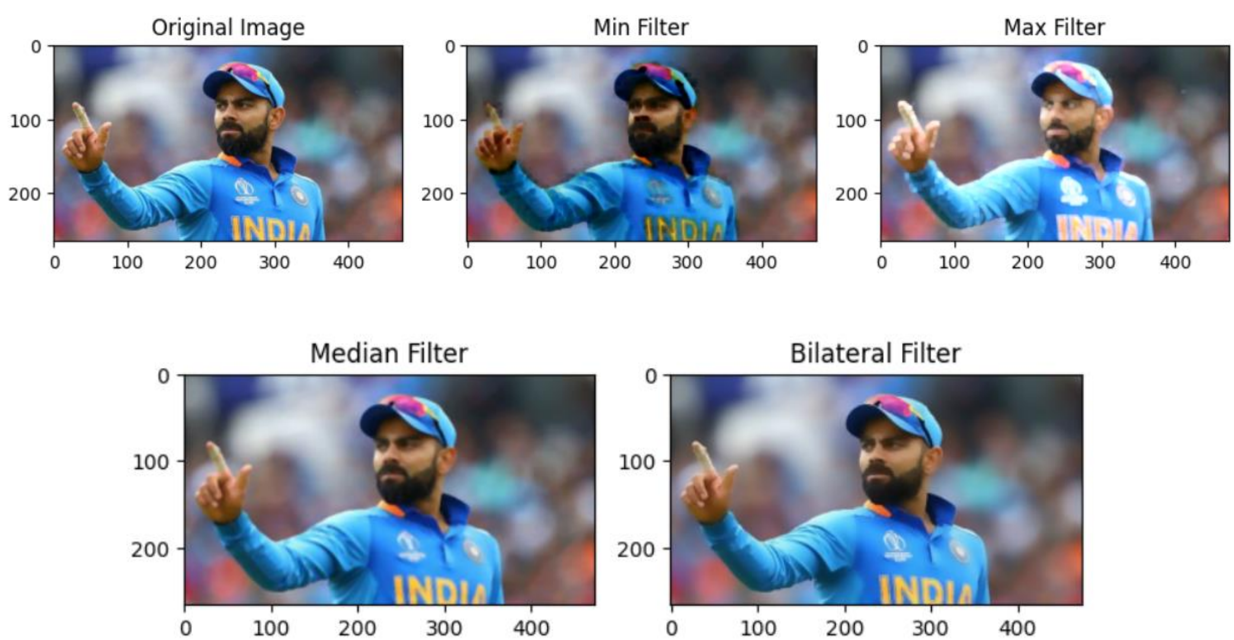
```
# Display the original image
axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axs[0].set_title("Original Image")

# Box Filter 5X5
kernel = np.ones((5, 5), np.float32) / 25
linear_filtered_image = apply_linear_filter(image, kernel)
axs[1].imshow(cv2.cvtColor(linear_filtered_image, cv2.COLOR_BGR2RGB))
axs[1].set_title("Box Filter")

# Gausian Filter
gaussian_filtered_image = apply_nonlinear_filter(image, "gaussian")
axs[2].imshow(cv2.cvtColor(gaussian_filtered_image, cv2.COLOR_BGR2RGB))
axs[2].set_title("Gaussian Filter")

plt.tight_layout()
```

output:


```
fig, axs = plt.subplots(2,3,figsize=(10,10))
# Display the original image
axs[0,0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axs[0,0].set_title("Original Image")
# Min Filter 5 x 5
min_filtered_image = apply_nonlinear_filter(image, "min")
axs[0,1].imshow(cv2.cvtColor(min_filtered_image, cv2.COLOR_BGR2RGB))
axs[0,1].set_title("Min Filter")
# Max Filter 5 x 5
max_filtered_image = apply_nonlinear_filter(image, "max")
axs[0,2].imshow(cv2.cvtColor(max_filtered_image, cv2.COLOR_BGR2RGB))
axs[0,2].set_title("Max Filter")
```

```
# Medain Filter 5 x 5
median_filtered_image = apply_nonlinear_filter(image, "median")
axs[1,0].imshow(cv2.cvtColor(median_filtered_image, cv2.COLOR_BGR2RGB))


axs[1,0].set_title("Median Filter")
# Biltaeral Filter
bil_filtered_image = apply_nonlinear_filter(image, "bilateral")
axs[1,1].imshow(cv2.cvtColor(bil_filtered_image, cv2.COLOR_BGR2RGB))
axs[1,1].set_title("Bilateral Filter")
axs[-1, -1].axis('off')
plt.tight_layout()
plt.show()
```

output:



Department of CSE-AIML & IoT

Experiment Number: 3

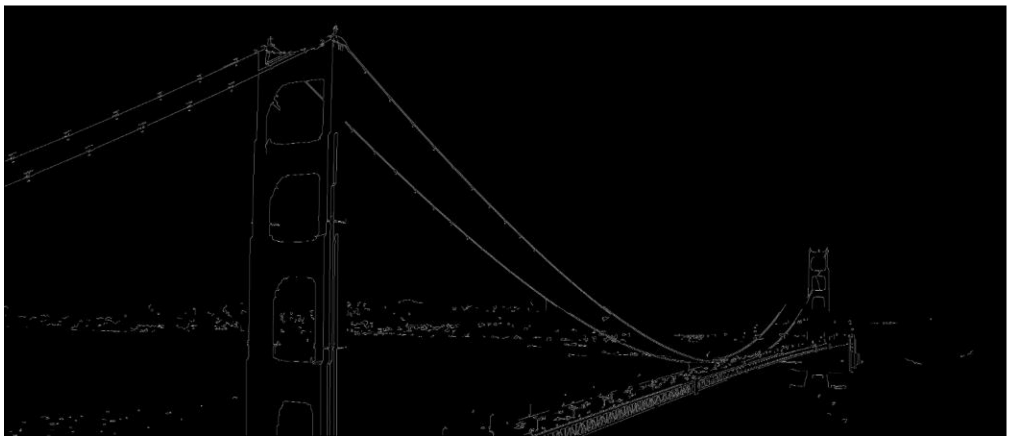
Title of the Experiment: Write a program to implement Canny edge detection algorithm using Python

Program:

```
import cv2
from google.colab.patches import cv2_imshow
# Load an image
image = cv2.imread('/content/bridge_2.jpg',cv2.IMREAD_GRAYSCALE)
# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(image, (5, 5), 0)
 # Apply Canny edge detection
edges = cv2.Canny(blurred, threshold1=100, threshold2=200)  # You can adjust the thresholds
print("Original image")
# Original Image
cv2_imshow(image)
print("Edge image")
# Edge Image
cv2_imshow(edges)
```

output:

Department of CSE-AIML & IoT



Department of CSE-AIML & IoT

Experiment Number: 4

Title of the Experiment: Write a program to extract interest points of an image by SIFT Detector using Python

Program

```
import cv2
from matplotlib import pyplot as plt

# Load the image
image_path = '/content/drive/MyDrive/Copy of download.jpg'
img = cv2.imread(image_path)

from google.colab import drive
drive.mount('/content/drive')

# Display the image using Matplotlib
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Turn off axis labels
plt.show()
```

output:

```
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Initialize the SIFT detector
sift = cv2.SIFT_create()

# Detect SIFT keypoints
keypoints = sift.detect(gray, None)

# Draw the keypoints on the image
output_image = cv2.drawKeypoints(img, keypoints, None)

# Display the image using Matplotlib
plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Turn off axis labels
plt.show()
```

output:

Department of CSE-AIML & IoT

Department of CSE-AIML & IoT

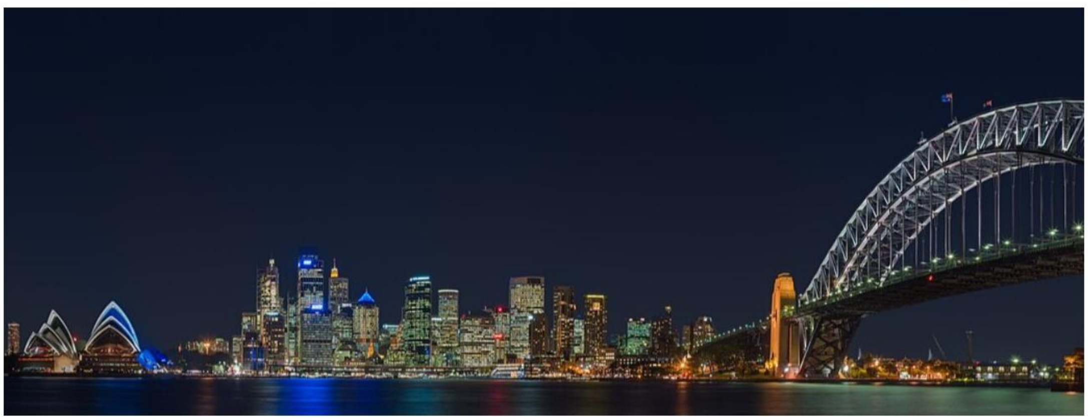
Experiment Number: 5

Title of the Experiment: Write a program to implement Image stiching panorama using Python

Program:

```
import cv2
import numpy as np

from google.colab import drive
drive.mount('/content/drive')

def stitch_images(images):

    stitcher = cv2.Stitcher_create()

    # Convert the list of images to a NumPy array
    images_np = np.array(images)

    # Stitch the images
    status, stitched_image = stitcher.stitch(images_np)

    # Check if the stitching was successful
    if status != cv2.Stitcher_OK:
        raise Exception("Image stitching failed with status code: {}".format(status))

    return stitched_image

image1 = cv2.imread("/content/drive/MyDrive/DLCV/IMAGES/1.jpeg")
image2 = cv2.imread("/content/drive/MyDrive/DLCV/IMAGES/2.jpeg")
```

```
# Stitch the images
stitched_image = stitch_images([image1, image2])

# Save the stitched image
cv2.imwrite("panorama.jpg", stitched_image)

from google.colab.patches import cv2_imshow
cv2_imshow(image1)
```



```
cv2_imshow(image2)
```

```
cv2_imshow(stitched_image)
```

Department of CSE-AIML & IoT

Department of CSE-AIML & IoT

Experiment Number: 6

Title of experiment: Write a program to implement Bag-of-words for a given image using Python.

Program:

```
!pip install opencv-python opencv-python-headless scikit-learn

import cv2
import numpy as np
from sklearn.cluster import MiniBatchKMeans

def extract_features(image_path, extractor):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    keypoints, descriptors = extractor.detectAndCompute(gray, None)
    return descriptors

def generate_codebook(descriptors, n_clusters):
    kmeans = MiniBatchKMeans(n_clusters=n_clusters)
    kmeans.fit(descriptors)
    return kmeans

def image_to_bow(features, kmeans):
    labels = kmeans.predict(features)
    histogram = np.bincount(labels, minlength=kmeans.n_clusters)
    return histogram / histogram.sum()  # normalize histogram

sift = cv2.SIFT_create()

# Extract features from your image
image_path = '/cric.jpg&width=500&resizemode=4'
descriptors = extract_features(image_path, sift)

# Generate codebook
n_clusters = 100  # Number of visual words
kmeans = generate_codebook(descriptors, n_clusters)
```

Department of CSE-AIML & IoT

```
# Represent image as bag-of-words
bow = image_to_bow(descriptors, kmeans)
print(bow)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set the value of `n_init` expl
  warnings.warn(
[0.00536836 0.00011422 0.00479726 0.00970874 0.01701885 0.01222159
 0.0005711  0.00959452 0.01210737 0.01382067 0.02101656 0.00639634
 0.01050828 0.01039406 0.01142204 0.01302113 0.00434038 0.01096516
 0.01690463 0.01050828 0.00662479 0.00628212 0.01827527 0.01382067
 0.00833809 0.00456882 0.0100514  0.00354083 0.01107938 0.01484866
 0.01382067 0.0094803  0.00970874 0.0106225  0.02147344 0.01233581
 0.00982296 0.00696745 0.00788121 0.00685323 0.01382067 0.0206739
 0.00799543 0.00868075 0.00548258 0.00970874 0.01073672 0.00571102
 0.01553398 0.01027984 0.00753855 0.00868075 0.00731011 0.00890919
 0.00731011 0.00970874 0.0100514  0.01290691 0.01324957 0.00651057
 0.01245003 0.01393489 0.00571102 0.00582524 0.01245003 0.01427756
 0.00925186 0.01085094 0.01941748 0.00788121 0.01165049 0.0111936
 0.00902342 0.01142204 0.01165049 0.00708167 0.01165049 0.00788121
 0.00548258 0.02044546 0.01050828 0.00456882 0.00970874 0.00845231
 0.00879497 0.00890919 0.00925186 0.00479726 0.00936608 0.00765277
 0.00890919 0.01027984 0.00890919 0.00456882 0.00685323 0.01302113
 0.01359223 0.00571102 0.01050828 0.00810965]
```

Department of CSE-AIML & IoT

Experiment Number: 7

Title of experiment: Write a program to implement Artificial Neural network to classify MNIST dataset.

Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

batch_size = 128
learning_rate = 0.001
num_epochs = 10

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

class SimpleANN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleANN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

   def forward(self, x):
    out = self.fc1(x)
    out = self.relu(out)
    out = self.fc2(out)
    return out
```

```
model = SimpleANN(28*28, 512, 10).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    train_correct = 0
    train_total = 0
    train_loss = 0.0

    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        train_loss += loss.item()

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

    train_accuracy = 100 * train_correct / train_total
    avg_train_loss = train_loss / len(train_loader)

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_train_loss:.4f}, Training Accuracy: {train_accuracy:.2f}%")
```

```
Epoch [1/10], Loss: 0.4526, Training Accuracy: 88.00%
Epoch [2/10], Loss: 0.1806, Training Accuracy: 94.59%
Epoch [3/10], Loss: 0.1221, Training Accuracy: 96.31%
Epoch [4/10], Loss: 0.0976, Training Accuracy: 97.01%
Epoch [5/10], Loss: 0.0823, Training Accuracy: 97.43%
Epoch [6/10], Loss: 0.0679, Training Accuracy: 97.88%
Epoch [7/10], Loss: 0.0596, Training Accuracy: 98.14%
Epoch [8/10], Loss: 0.0538, Training Accuracy: 98.27%
Epoch [9/10], Loss: 0.0472, Training Accuracy: 98.46%
Epoch [10/10], Loss: 0.0417, Training Accuracy: 98.65%
```

```
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f'Test Accuracy of the model: {100 * correct / total} %')
```

```
Test Accuracy of the model: 97.62 %
```

Department of CSE-AIML & IoT

Experiment Number: 8

Title of experiment: Classification CIFAR-10 Dataset using AlexNet using PyTorch

Program:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import time
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
# Download the CIFAR-10 dataset
train_data = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(train_data, batch_size=4, shuffle=True, num_workers=2)

test_data = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(test_data, batch_size=4, shuffle=False, num_workers=2)

classes = ('Airplane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck

# Define the AlexNet model
model = torchvision.models.alexnet()

model.classifier[1] = nn.Linear(9216,4096)
model.classifier[4] = nn.Linear(4096,1024)
model.classifier[6] = nn.Linear(1024,10)

model.eval()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# move the input and model to GPU for speed if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model.to(device)

for epoch in range(10):  # loop over the dataset multiple times
  running_loss = 0.0
  start_time = time.time()
  for i, data in enumerate(trainloader, 0):
      # get the inputs; data is a list of [inputs, labels]
      inputs, labels = data[0].to(device), data[1].to(device)

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      output = model(inputs)
      loss = criterion(output, labels)
      loss.backward()
      optimizer.step()

      #Time
      end_time = time.time()
      time_taken = end_time - start_time

      # print statistics
      running_loss += loss.item()
      if i % 2000 == 1999:    # print every 2000 mini-batches
          print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss / 2000))
          print('Time:',time_taken)
          running_loss = 0.0

print('Finished Training of AlexNet')
```

```
Time: 72.44327712059021
[9,  6000] loss: 0.134
Time: 108.67101240158081
[9,  8000] loss: 0.154
Time: 145.1121928691864
[9, 10000] loss: 0.188
Time: 181.64948153495789
[9, 12000] loss: 0.186
Time: 217.91190910339355
[10,  2000] loss: 0.109
Time: 37.95023965835571
[10,  4000] loss: 0.126
Time: 74.50496745109558
[10,  6000] loss: 0.123
Time: 110.27510452270508
[10,  8000] loss: 0.140
Time: 146.8265779018402
[10, 10000] loss: 0.149
Time: 183.3646821975708
[10, 12000] loss: 0.160
Time: 220.76011538505554
Finished Training of AlexNet
```

```
#Testing Accuracy
correct = 0
total = 0
with torch.no_grad():
for data in testloader:
    images, labels = data[0].to(device), data[1].to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %.2f %%' % (100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 77.39 %
```

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE, EEE, ME, ECE, CSE, EIE, IT Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 135th Rank in Engineering Category
Recognized as "College with Potential for Excellence" by UGC
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India.
Telephone No: 040-2304 2758/59/60, Fax: 040-23042761
E-mail: postbox@vnrvjiet.ac.in, Website: www.vnrvjiet.ac.in

Estd.1995

Department of CSE-AIML & IoT

Experiment Number: 9

Title of experiment: Classification CIFAR-10 Dataset using ZFNet using PyTorch

Program:

```
pip install torchvision

import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=6,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=6,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

"""### Displaying some images"""

import matplotlib.pyplot as plt
import numpy as np

def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

Department of CSE-AIML & IoT

```
# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))

"""### defining ZF-net network"""

import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

"""### Defining a Loss function and optimizer"""

import torch.optim as optim

criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# finally we will train our neural network

for epoch in range(20):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[16,  2000] loss: 0.578
[16,  4000] loss: 0.638
[16,  6000] loss: 0.654
[16,  8000] loss: 0.679
[17,  2000] loss: 0.583
[17,  4000] loss: 0.620
[17,  6000] loss: 0.628
[17,  8000] loss: 0.661
[18,  2000] loss: 0.544
[18,  4000] loss: 0.589
[18,  6000] loss: 0.616
[18,  8000] loss: 0.661
[19,  2000] loss: 0.529
[19,  4000] loss: 0.578
[19,  6000] loss: 0.629
[19,  8000] loss: 0.613
[20,  2000] loss: 0.499
[20,  4000] loss: 0.573
[20,  6000] loss: 0.602
[20,  8000] loss: 0.609
Finished Training
```

```
# now we will save our model

PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)

#let's test the accuracy of model
dataiter = iter(testloader)
images, labels = next(dataiter)

# print images
imshow(torchvision.utils.make_grid(images))


print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

# We will now see how the neural network predicts the above image

outputs = net(images)
```

Department of CSE-AIML & IoT

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))

# Let us look at how the network performs on the whole dataset.

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
```

```
Accuracy of the network on the 10000 test images: 61 %
```

```
    100 * correct / total))

# Let's see which classes performed well individually

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
```

```
classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 59 %
Accuracy of   car : 79 %
Accuracy of  bird : 52 %
Accuracy of   cat : 45 %
Accuracy of  deer : 60 %
Accuracy of   dog : 43 %
Accuracy of  frog : 66 %
Accuracy of horse : 61 %
Accuracy of  ship : 78 %
Accuracy of truck : 72 %
```

Department of CSE-AIML & IoT

Experiment Number: 10

Title of experiment: Classification CIFAR-10 Dataset using VGGNet using PyTorch

Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.utils.data import random_split

# Define data transformations
transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.7),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

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

# Load CIFAR-10 dataset and split into train and validation sets
torch.manual_seed(2021)
train = torchvision.datasets.CIFAR10("data/", train=True, download=True, transform=transform_train)
val_size = 10000
train_size = len(train) - val_size
train, val = random_split(train, [train_size, val_size])
test = torchvision.datasets.CIFAR10("data/", train=False, download=True, transform=transform_test)

# Create data loaders
BATCH_SIZE = 64
train_loader = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
```

```
val_loader = DataLoader(val, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test, batch_size=8, shuffle=False)

# Define the VGG16-based model
import torch.nn.functional as F

class VGG16_NET(nn.Module):
    def __init__(self):
        super(VGG16_NET, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)

        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)

        self.conv5 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)
        self.conv6 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.conv7 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)

        self.conv8 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1)
        self.conv9 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv10 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)

        self.conv11 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv12 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv13 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)

        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc14 = nn.Linear(25088, 4096)
        self.fc15 = nn.Linear(4096, 4096)
        self.fc16 = nn.Linear(4096, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.maxpool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.maxpool(x)
```

```
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = F.relu(self.conv7(x))
        x = self.maxpool(x)
        x = F.relu(self.conv8(x))
        x = F.relu(self.conv9(x))
        x = F.relu(self.conv10(x))
        x = self.maxpool(x)
        x = F.relu(self.conv11(x))
        x = F.relu(self.conv12(x))
        x = F.relu(self.conv13(x))
        x = self.maxpool(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc14(x))
        x = F.dropout(x, 0.5)  # Dropout was included to combat overfitting
        x = F.relu(self.fc15(x))
        x = F.dropout(x, 0.5)
        x = self.fc16(x)
        return x

# Initialize the model and move it to GPU if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = VGG16_NET()
model = model.to(device=device)
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

# Training loop
num_epochs = 5

for epoch in range(num_epochs):
    loss_var = 0

    for idx, (images, labels) in enumerate(train_loader):
        images = images.to(device=device)
        labels = labels.to(device=device)

        optimizer.zero_grad()
        scores = model(images)
```

```
        loss = criterion(scores, labels)
        loss.backward()
        optimizer.step()
        loss_var += loss.item()

        if idx % 64 == 0:
            print(f'Epoch [{epoch + 1}/{num_epochs}] || Step [{idx + 1}/{len(train_loader)}] ||
Loss:{loss_var / len(train_loader)}')

    print(f"Loss at epoch {epoch + 1} || {loss_var / len(train_loader)}")

    with torch.no_grad():
        correct = 0
        samples = 0
        for idx, (images, labels) in enumerate(val_loader):
            images = images.to(device=device)
            labels = labels.to(device=device)
            outputs = model(images)
            _, preds = outputs.max(1)
            correct += (preds == labels).sum()
            samples += preds.size(0)

        print(f"Validation accuracy {float(correct) / float(samples) * 100:.2f} percentage || Correct
{correct} out of {samples} samples")
```

```
Epoch [3/5] || Step [193/625] || Loss:0.16004273042678832
Epoch [3/5] || Step [257/625] || Loss:0.21844405760765076
Epoch [3/5] || Step [321/625] || Loss:0.27165847091674805
Epoch [3/5] || Step [385/625] || Loss:0.32802225918769834
Epoch [3/5] || Step [449/625] || Loss:0.3839901364803314
Epoch [3/5] || Step [513/625] || Loss:0.4378532968997955
Epoch [3/5] || Step [577/625] || Loss:0.49182252688407896
Loss at epoch 3 || 0.5326728787899018
Validation accuracy 76.64 percentage || Correct 7664 out of 10000 samples
Epoch [4/5] || Step [1/625] || Loss:0.0006602890014648438
Epoch [4/5] || Step [65/625] || Loss:0.04323548884391785
Epoch [4/5] || Step [129/625] || Loss:0.08637202672958375
Epoch [4/5] || Step [193/625] || Loss:0.12967704374790193
Epoch [4/5] || Step [257/625] || Loss:0.17452217032909392
Epoch [4/5] || Step [321/625] || Loss:0.2195039835214615
Epoch [4/5] || Step [385/625] || Loss:0.2616340528488159
Epoch [4/5] || Step [449/625] || Loss:0.30644493505954745
Epoch [4/5] || Step [513/625] || Loss:0.3492969330072403
Epoch [4/5] || Step [577/625] || Loss:0.39412975077629087
Loss at epoch 4 || 0.427018733382225
Validation accuracy 79.06 percentage || Correct 7906 out of 10000 samples
Epoch [5/5] || Step [1/625] || Loss:0.000799348783493042
Epoch [5/5] || Step [65/625] || Loss:0.03444554101228714
Epoch [5/5] || Step [129/625] || Loss:0.06952987838983536
Epoch [5/5] || Step [193/625] || Loss:0.10299175993204117
Epoch [5/5] || Step [257/625] || Loss:0.13587589260339736
Epoch [5/5] || Step [321/625] || Loss:0.17082266265153884
Epoch [5/5] || Step [385/625] || Loss:0.20424372217655182
Epoch [5/5] || Step [449/625] || Loss:0.2354231837749481
Epoch [5/5] || Step [513/625] || Loss:0.2719827586889267
Epoch [5/5] || Step [577/625] || Loss:0.3041187573194504
Loss at epoch 5 || 0.3293868698358536
Validation accuracy 78.25 percentage || Correct 7825 out of 10000 samples
```

```python
# Test the model on the test dataset
correct = 0
samples = 0
for idx, (images, labels) in enumerate(test_loader):
    images = images.to(device=device)
    labels = labels.to(device=device)

    outputs = model(images)
    _, preds = outputs.max(1)
    correct += (preds == labels).sum()
    samples += preds.size(0)

print(f"Test accuracy {float(correct) / float(samples) * 100:.2f} percentage || Correct {correct} out of {samples} samples")
```

```
Test accuracy 78.13 percentage || Correct 7813 out of 10000 samples
```

Department of CSE-AIML & IoT

Experiment Number: 11

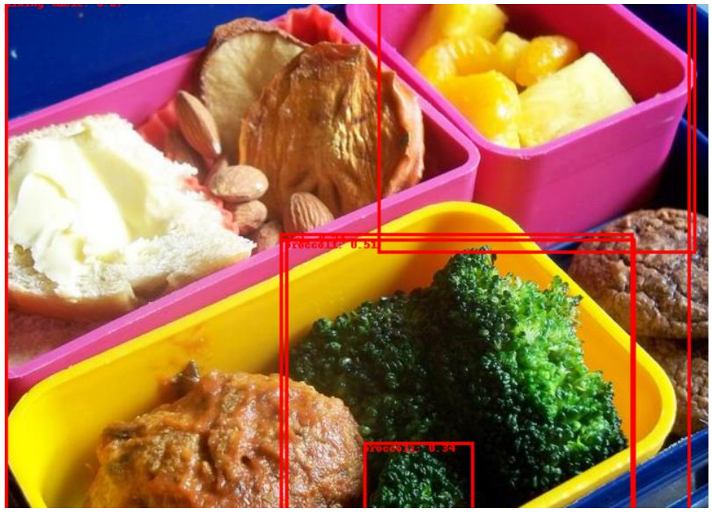
Title of experiment: Object detection for PASCAL VOC dataset using YOLO Object Detection using PyTorch

Program:

```
import torch
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt

model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

for idx in range(1,11):

    # Load the image
    image_path = "/content/"+str(idx)+".jpg"
    img = Image.open(image_path)

    # Perform object detection
    results = model(img)

    # Get detected labels, confidences, and bounding boxes
    labels = results.names
    detected_labels = [labels[int(label)] for label in results.pred[0][:, -1].cpu().numpy().astype(int)]
    confidences = results.pred[0][:, -2].cpu().numpy()
    boxes = results.pred[0][:, :-2].cpu().numpy()

    # Create a copy of the original image for visualization
    detected_img = img.copy()
    draw = ImageDraw.Draw(detected_img)

    # Draw bounding boxes on the image
    for label, confidence, box in zip(detected_labels, confidences, boxes):
        x1, y1, x2, y2 = box
        draw.rectangle([x1, y1, x2, y2], outline="red", width=3)
        draw.text((x1, y1), f"{label}: {confidence:.2f}", fill="red")

    # Display the image with bounding boxes
```

```
plt.figure(figsize=(5,5))

 display(img)
display(detected_img)

# Calculate confidences
labels = results.names
detected_labels = [labels[int(label)] for label in results.pred[0][:, -1].cpu().numpy().astype(int)]
confidences = results.pred[0][:, -2].cpu().numpy()
print("Detected Labels:")
for i, (label, confidence) in enumerate(zip(detected_labels, confidences)):
   print(f"{i+1}. {label}: {confidence:.2f}")
```

output



```
Detected Labels:
1. bowl: 0.78
2. broccoli: 0.51
3. bowl: 0.34
4. broccoli: 0.34
5. dining table: 0.27
```

Department of CSE-AIML & IoT
Experiment Number: 12

Title of experiment: Classification CIFAR-10 Dataset using VGGNet using PyTorch

Program:

```
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from tensorflow.keras.datasets.cifar10 import load_data
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import
Dense,Reshape,Flatten,Conv2D,Conv2DTranspose,LeakyReLU,Dropout
from matplotlib import pyplot

# define the standalone discriminator model
def define_discriminator(in_shape=(32, 32, 3)):
    model = Sequential()
    # normal
    model.add(Conv2D(64, (3, 3), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(128, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(128, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample
    model.add(Conv2D(256, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # classifier
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

```
    return model

# define the standalone generator model
def define_generator(latent_dim):
    model = Sequential()
    # foundation for 4x4 image
    n_nodes = 256 * 4 * 4
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((4, 4, 256)))
    # upsample to 8x8
    model.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 16x16
    model.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 32x32
    model.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # output layer
    model.add(Conv2D(3, (3, 3), activation='tanh', padding='same'))
    return model

# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model
```

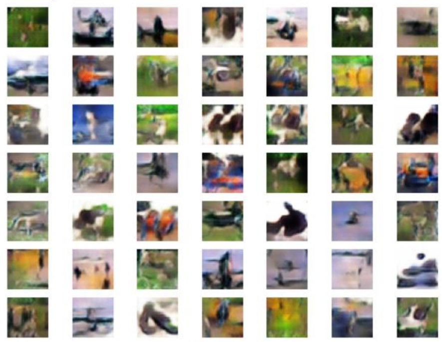
Department of CSE-AIML & IoT

```
# load and prepare cifar10 training images
def load_real_samples():
    # load cifar10 dataset
    (trainX, _), (_, _) = load_data()
    # convert from unsigned ints to floats
    X = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5
    return X

# select real samples
def generate_real_samples(dataset, n_samples):
    # choose random instances
    ix = randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    X = dataset[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, 1))
    return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = zeros((n_samples, 1))
    return X, y
```

```
# create and save a plot of generated images
def save_plot(examples, epoch, n=7):
    # scale from [-1,1] to [0,1]
    examples = (examples + 1) / 2.0
    # plot images
    for i in range(n * n):
        # define subplot
        pyplot.subplot(n, n, 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(examples[i])
    # save plot to file
    # filename = 'generated_plot_e%03d.png' % (epoch+1)
    # pyplot.savefig(filename)
    pyplot.show()

# evaluate the discriminator, plot generated images, save generator model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=150):
    # prepare real samples
    X_real, y_real = generate_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real * 100, acc_fake * 100))
    # save plot
    save_plot(x_fake, epoch)
    g_model.save(filename)

# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch=128):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_epochs):
        # enumerate batches over the training set
```

```
    for j in range(bat_per_epo):
        # get randomly selected 'real' samples
        X_real, y_real = generate_real_samples(dataset, half_batch)
        # update discriminator model weights
        d_loss1, _ = d_model.train_on_batch(X_real, y_real)
        # generate 'fake' examples
        X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
        # update discriminator model weights
        d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
        # prepare points in latent space as input for the generator
        X_gan = generate_latent_points(latent_dim, n_batch)
        # create inverted labels for the fake samples
        y_gan = ones((n_batch, 1))
        # update the generator via the discriminator's error
        g_loss = gan_model.train_on_batch(X_gan, y_gan)
        # summarize loss on this batch
        if j % 100 == 0:
            print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' % (i + 1, j + 1, bat_per_epo, d_loss1,
d_loss2, g_loss))
    # evaluate the model performance, sometimes
    # if (i+1) % 10 == 0:
    summarize_performance(i, g_model, d_model, dataset, latent_dim)

# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim, 20, 256)

from google.colab import drive
drive.mount('/content/drive')
```

```
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 7ms/step
5/5 [==============================] - 0s 6ms/step
>Accuracy real: 61%, fake: 75%
```



```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```