MACHINE INTELLIGENCE LAB

Submitted by:

Christy Binu
Roll number: 97422607009

III Semester

Msc. Computer Science
With Specialization in
Artificial Intelligence

Department of Computer Science

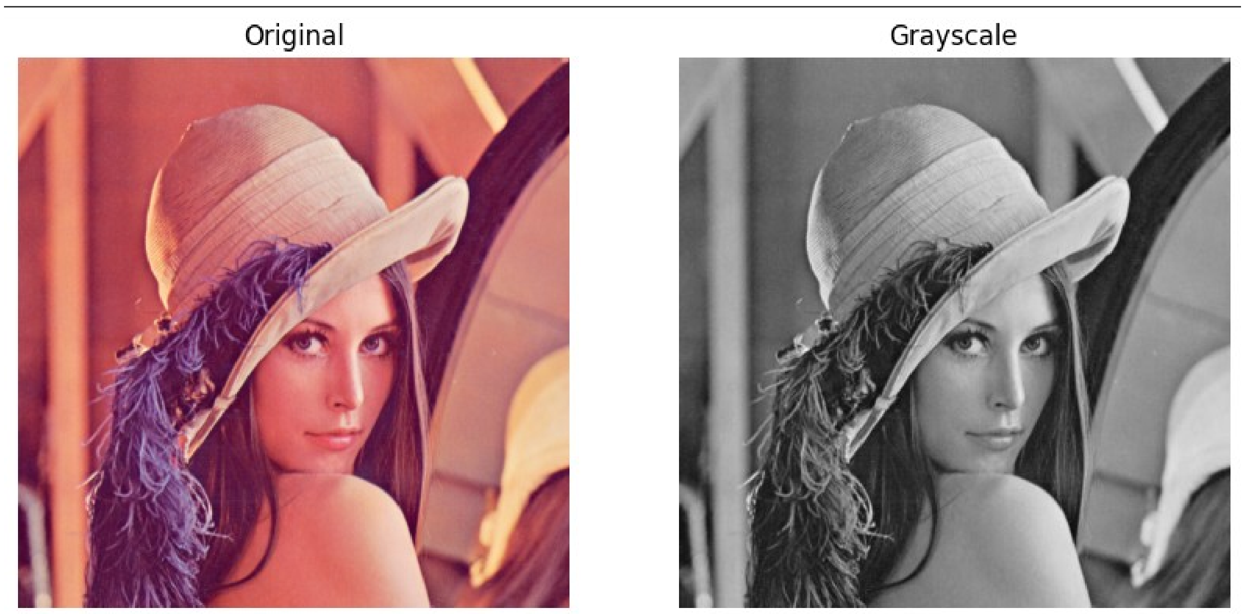
CYCLE – 1

1. Read an image and convert it into gray scale image without using builtin function for the function

Code:

```
rom PIL import Image
import matplotlib.pyplot as plt

def convert_to_grayscale(image_path):
img = Image.open(image_path)
width, height = img.size
grayscale_img = Image.new('L', (width, height))
for y in range(height):
for x in range(width):
pixel = img.getpixel((x, y))
gray_value = int(0.299 * pixel[0] + 0.587 * pixel[1] + 0.114 * pixel[2])
grayscale_img.putpixel((x, y), gray_value)
grayscale_img.save("grayscale_image.jpg")

# Display both images side by side
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
# Plot original image
original_img = plt.imread(image_path)
axes[0].imshow(original_img)
axes[0].set_title('Original')
axes[0].axis('off') # Hide axes
# Plot grayscale image
grayscale_array = plt.imread("grayscale_image.jpg")
axes[1].imshow(grayscale_array, cmap='gray')
axes[1].set_title('Grayscale')
axes[1].axis('off') # Hide axes
plt.show()

image_path = "test1.png"
convert_to_grayscale(image_path)
```

Output:

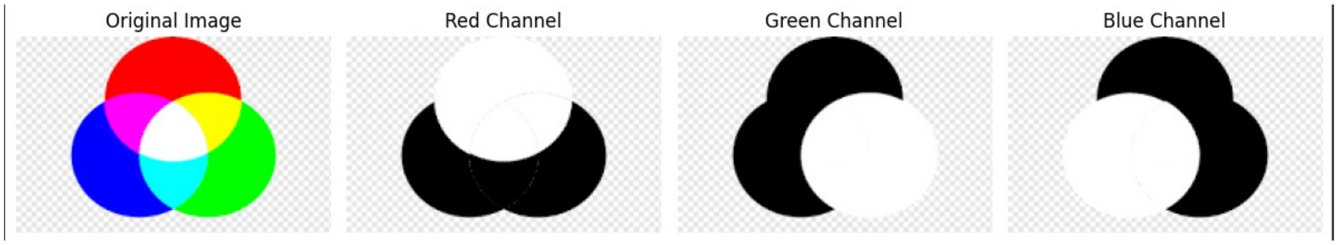


2. Read an image and display the RGB channel images separately.

Code:

```
from PIL import Image
import matplotlib.pyplot as plt

def display_rgb_channels(image_path):
    img = Image.open(image_path)

    # Convert the image to RGB mode if it's not already in RGB
    if img.mode != 'RGB':
        img = img.convert('RGB')

    # Split the image into RGB channels
    r, g, b = img.split()

    # Display original image
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 4, 1)
    plt.title('Original Image')
    plt.imshow(img)
    plt.axis('off')

    # Display each channel separately
    plt.subplot(1, 4, 2)
    plt.title('Red Channel')
    plt.imshow(r, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 4, 3)
    plt.title('Green Channel')
    plt.imshow(g, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 4, 4)
    plt.title('Blue Channel')
    plt.imshow(b, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# Provide the path to the image file
image_path = "rgb.png"  # Replace with your image file path
display_rgb_channels(image_path)
```

Output:

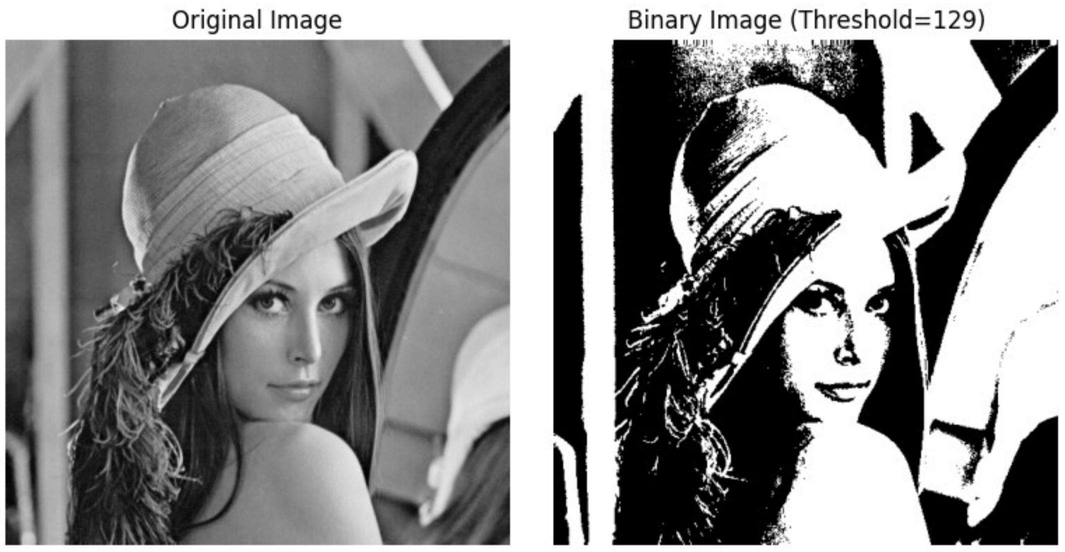


3. Read an image and convert it into binary image using threshold.

Code:

```
from PIL import Image
import matplotlib.pyplot as plt

def convert_to_binary(image_path, threshold_value):

img = Image.open(image_path)
img = img.convert('L')

binary_img = img.point(lambda p: 0 if p < threshold_value else 255, '1')
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.title('Binary Image (Threshold={})'.format(threshold_value))
plt.imshow(binary_img, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()


image_path = "test1.png"
threshold = 129

convert_to_binary(image_path, threshold)
```

Output:



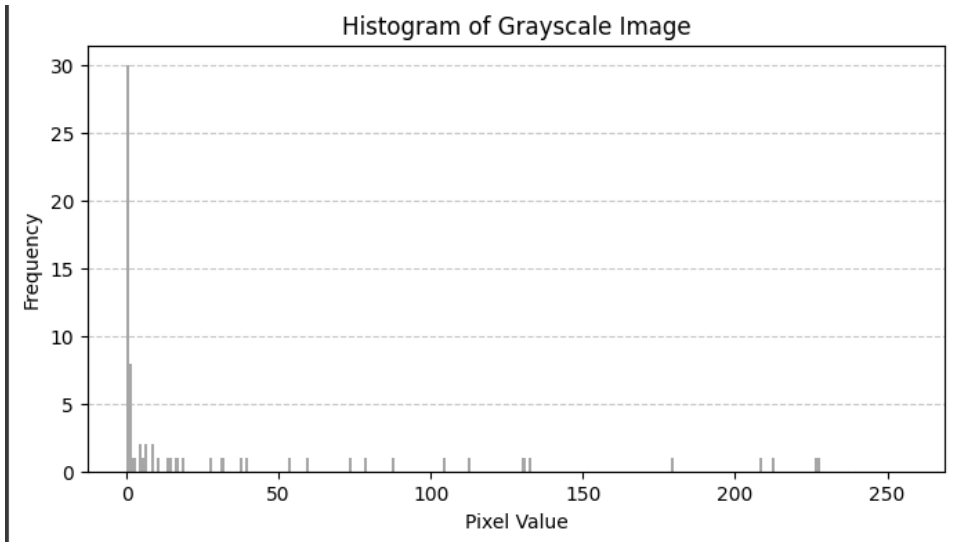
4. Display the histogram of the gray scale image.

Code:

```
from PIL import Image
import matplotlib.pyplot as plt

def display_histogram(image_path):
# Open the image
img = Image.open(image_path)
# Convert the image to grayscale if it's not already in grayscale
img = img.convert('L')
```

```
# Calculate the histogram
histogram = img.histogram()
# Display the histogram
plt.figure(figsize=(8, 4))
plt.title('Histogram of Grayscale Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.hist(histogram, bins=256, range=(0, 256), color='gray', alpha=0.7)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Provide the path to the image file
image_path = "grayscale_image.jpg" # Replace with your image file path

display_histogram(image_path)
```

Output:



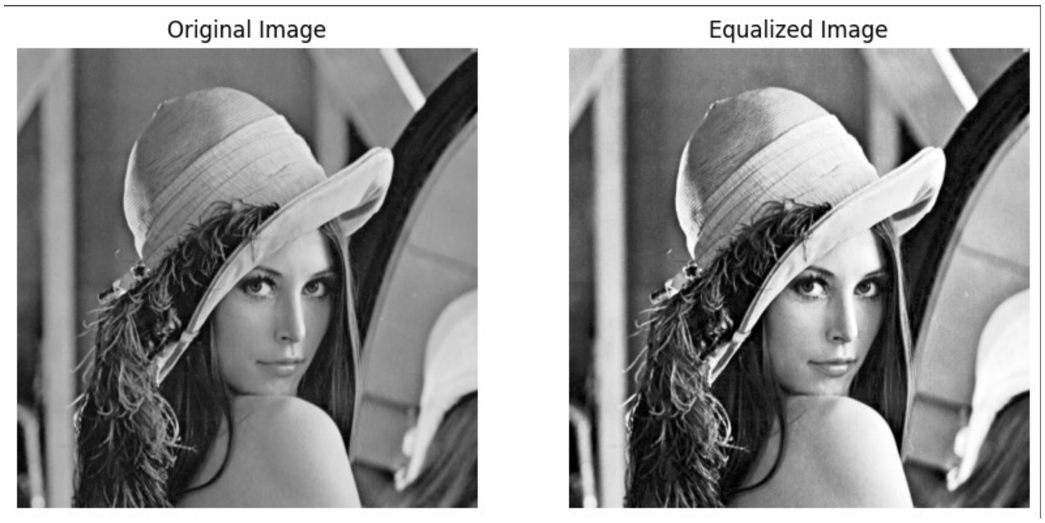
5. Apply histogram equalization on an image and display the resultant image.

Code:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

path = "test1.png"
img = cv.imread(path)

# Convert image to grayscale
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

```
# Perform histogram equalization
equ = cv.equalizeHist(gray_img)

# Plot original and equalized images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equ, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```

Output:



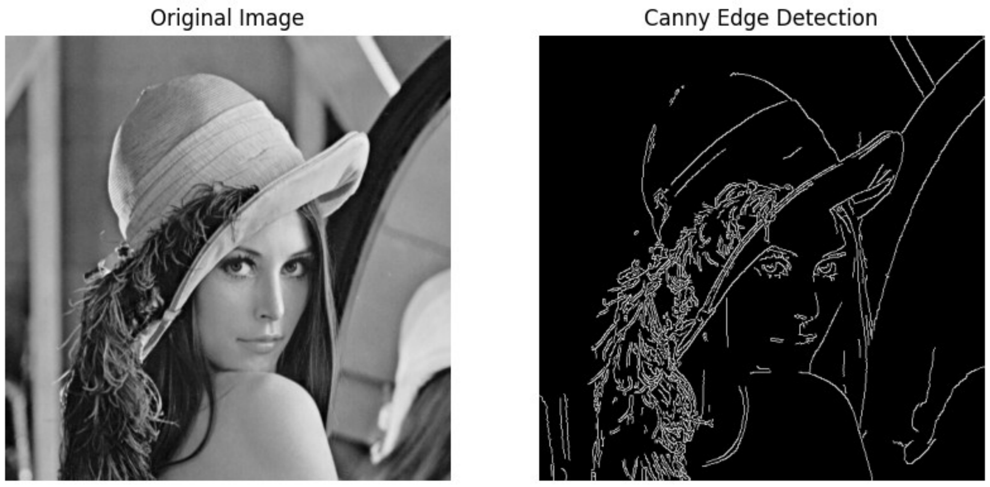
6. Display the edge map of an image with any edge detection algorithm

Code:

```
import cv2
import matplotlib.pyplot as plt

# Read the image
image_path = "test1.png" # Replace with your image file path
img = cv2.imread(image_path, 0) # Read image in grayscale

# Apply Canny edge detection
```

```
edges = cv2.Canny(img, 100, 200) # Adjust threshold values for best results

# Plot the original and edge-detected images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')

plt.show()
```

Output:



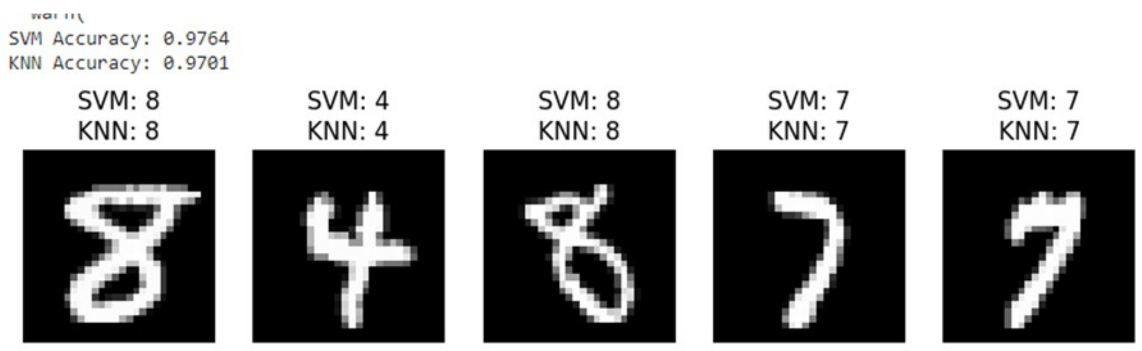
7. Download any OCR dataset and perform the classification with SVM and KNN. Compare the obtained result

Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm, neighbors, metrics
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt

mnist = fetch_openml('mnist_784', version=1)
X = np.array(mnist.data.astype('int'))
```

```
y = np.array(mnist.target.astype('int'))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_classifier = svm.SVC()
svm_classifier.fit(X_train, y_train)
svm_predictions = svm_classifier.predict(X_test)

knn_classifier = neighbors.KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)

svm_accuracy = metrics.accuracy_score(y_test, svm_predictions)
knn_accuracy = metrics.accuracy_score(y_test, knn_predictions)

print(f"SVM Accuracy: {svm_accuracy:.4f}")
print(f"KNN Accuracy: {knn_accuracy:.4f}")

fig, axes = plt.subplots(1, 5, figsize=(10, 2))

for i, ax in enumerate(axes):
    ax.imshow(X_test[i].reshape(28, 28), cmap='gray')
    ax.set_title(f'SVM: {svm_predictions[i]}\nKNN: {knn_predictions[i]}')
    ax.axis('off')

plt.show()
```

Output:



```
  warn(
SVM Accuracy: 0.9764
KNN Accuracy: 0.9701
```

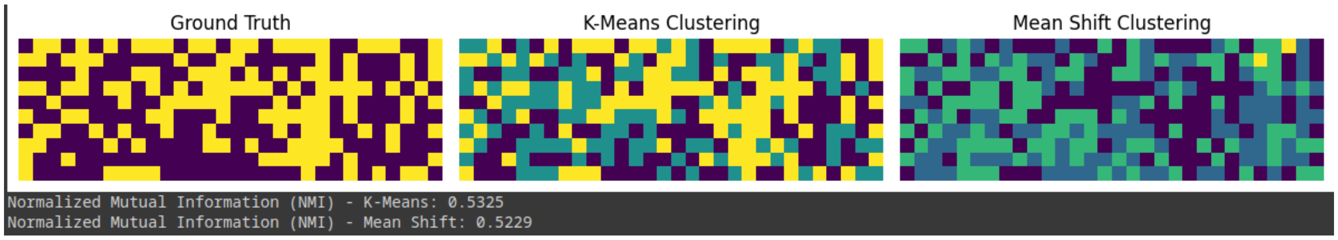
8. Implement any two segmentation algorithms and compare the efficiency with ground truth

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import normalized_mutual_info_score

# Create a synthetic ground truth image
np.random.seed(42)
ground_truth, _ = make_blobs(n_samples=300, centers=3, random_state=42, cluster_std=2.0)
ground_truth = StandardScaler().fit_transform(ground_truth)

# Generate image with ground truth labels
ground_truth_labels = np.argmax(ground_truth, axis=1)
ground_truth_labels = ground_truth_labels.reshape((10, 30))

# Plot ground truth
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.imshow(ground_truth_labels, cmap='viridis')
plt.title('Ground Truth')
plt.axis('off')

# Generate image data for clustering
data, _ = make_blobs(n_samples=300, centers=3, random_state=42, cluster_std=2.0)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(data)
kmeans_labels = kmeans_labels.reshape((10, 30))

# Plot K-Means clustering result
plt.subplot(1, 3, 2)
plt.imshow(kmeans_labels, cmap='viridis')
plt.title('K-Means Clustering')
plt.axis('off')

# Apply Mean Shift clustering
bandwidth = estimate_bandwidth(data, quantile=0.2, n_samples=300)
meanshift = MeanShift(bandwidth=bandwidth, bin_seeding=True)
meanshift_labels = meanshift.fit_predict(data)
meanshift_labels = meanshift_labels.reshape((10, 30))
```

```
# Plot Mean Shift clustering result
plt.subplot(1, 3, 3)
plt.imshow(meanshift_labels, cmap='viridis')
plt.title('Mean Shift Clustering')
plt.axis('off')

plt.tight_layout()
plt.show()

# Compare clustering results with ground truth
nmi_kmeans = normalized_mutual_info_score(ground_truth_labels.flatten(), kmeans_labels.flatten())
nmi_meanshift = normalized_mutual_info_score(ground_truth_labels.flatten(),
meanshift_labels.flatten())

print(f"Normalized Mutual Information (NMI) - K-Means: {nmi_kmeans:.4f}")
print(f"Normalized Mutual Information (NMI) - Mean Shift: {nmi_meanshift:.4f}")
```

Output:



```
Normalized Mutual Information (NMI) - K-Means: 0.5325
Normalized Mutual Information (NMI) - Mean Shift: 0.5229
```

9. Input an image and perform the following morphological operations

i) Dilation
ii) Erosion
iii) Opening
iv) Closing

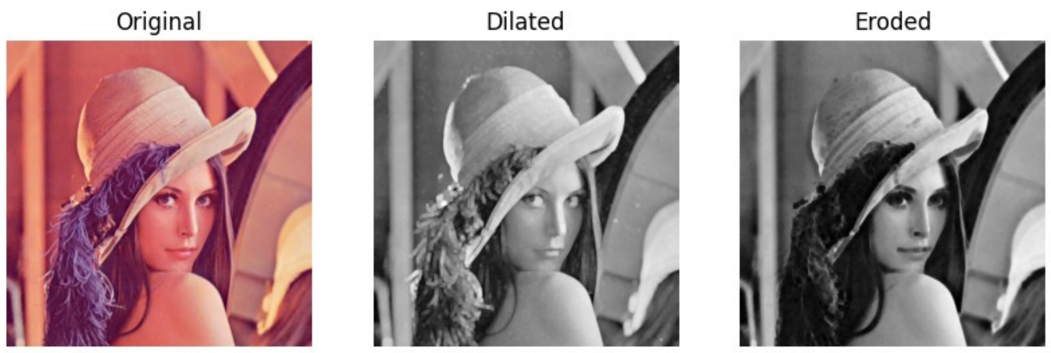
Display the results.

Code:

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

# Read the image
image = cv2.imread('test1.png')  # Replace with your image file

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Define the kernel for morphological operations
```

```
kernel = np.ones((5, 5), np.uint8)  # You can adjust the size of the kernel as needed

# Perform morphological operations
dilated_image = cv2.dilate(gray_image, kernel, iterations=1)
eroded_image = cv2.erode(gray_image, kernel, iterations=1)
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)
closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel)

# Display the original and processed images
plt.figure(figsize=(10, 10))

plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(dilated_image, cmap='gray')
plt.title('Dilated')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(eroded_image, cmap='gray')
plt.title('Eroded')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(opened_image, cmap='gray')
plt.title('Opened')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(closed_image, cmap='gray')
plt.title('Closed')
plt.axis('off')

plt.show()
```

Output:

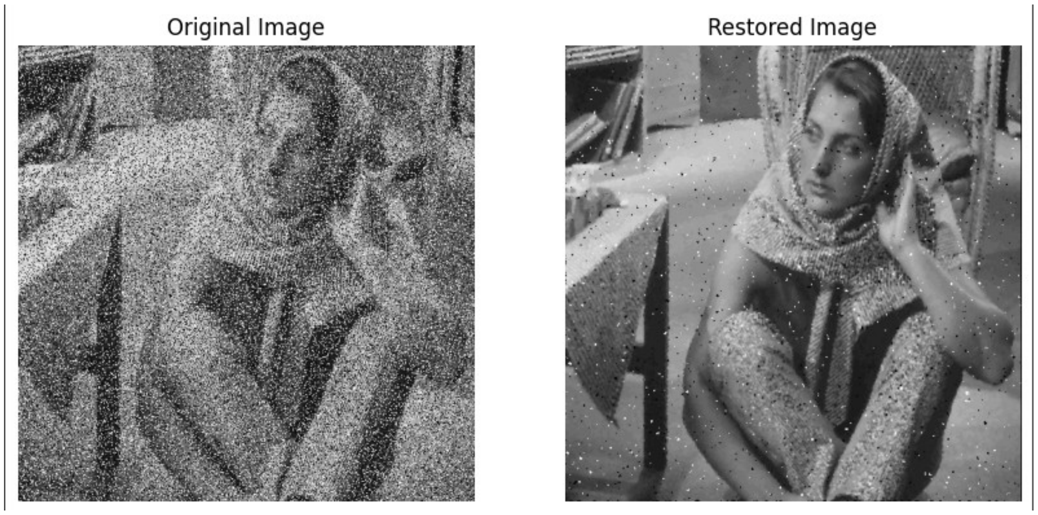


10. Implement any image restoration algorithm

Code:

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

def median_filter(data, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = np.zeros_like(data)
    for i in range(len(data)):
        for j in range(len(data[0])):
            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k - indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
    return data_final

img = cv2.imread("test2.png", cv2.IMREAD_GRAYSCALE)
removed_noise = median_filter(img, 3)

# Plot original and restored images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(removed_noise, cmap='gray')
plt.title('Restored Image')
plt.axis('off')

plt.show()
```

Output:



CYCLE – 2

I Build a small Convolutional Neural Network (CNN) model using any of deep libraries for:

a) Image Recognition/ Classification

b) Digit Identification

Code:

a)

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

# Train the model
model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=(test_images,
test_labels))
```

b)

```
import tensorflow as tf
from tensorflow.keras import layers, models
```

```
# Load MNIST dataset
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define the CNN model for digit identification
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])

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

# Reshape the input data to fit the CNN input shape
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images,
test_labels))
```

Output:

a)

```
Epoch 6/10

782/782 [==============================] - 4s 5ms/step - loss: 0.8482 - accuracy: 0.7076 -
val_loss: 0.9099 - val_accuracy: 0.6842
Epoch 7/10
782/782 [==============================] - 4s 5ms/step - loss: 0.7968 - accuracy: 0.7231 -
val_loss: 0.8801 - val_accuracy: 0.6972
Epoch 8/10
782/782 [==============================] - 4s 6ms/step - loss: 0.7487 - accuracy: 0.7395 -
val_loss: 0.8674 - val_accuracy: 0.6994
Epoch 9/10
```

```
782/782 [==============================] - 4s 5ms/step - loss: 0.7155 - accuracy: 0.7491 -
val_loss: 0.8860 - val_accuracy: 0.6953
Epoch 10/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6783 - accuracy: 0.7652 -
val_loss: 0.8507 - val_accuracy: 0.7162
<keras.src.callbacks.History at 0x7e817c7c0b50>
```

b)

```
Epoch 1/5 938/938 [==============================] - 7s 5ms/step - loss: 0.1821
- accuracy: 0.9449 - val_loss: 0.0657 - val_accuracy: 0.9812 Epoch 2/5 938/938
[==============================] - 5s 5ms/step - loss: 0.0527 - accuracy:
0.9838 - val_loss: 0.0462 - val_accuracy: 0.9854 Epoch 3/5 938/938
[==============================] - 5s 5ms/step - loss: 0.0392 - accuracy:
0.9878 - val_loss: 0.0293 - val_accuracy: 0.9895 Epoch 4/5 938/938
[==============================] - 5s 5ms/step - loss: 0.0287 - accuracy:
0.9906 - val_loss: 0.0390 - val_accuracy: 0.9862 Epoch 5/5 938/938
[==============================] - 5s 6ms/step - loss: 0.0237 - accuracy:
0.9924 - val_loss: 0.0313 - val_accuracy: 0.9891
```

II How to use Pre-trained CNN models for feature extraction.

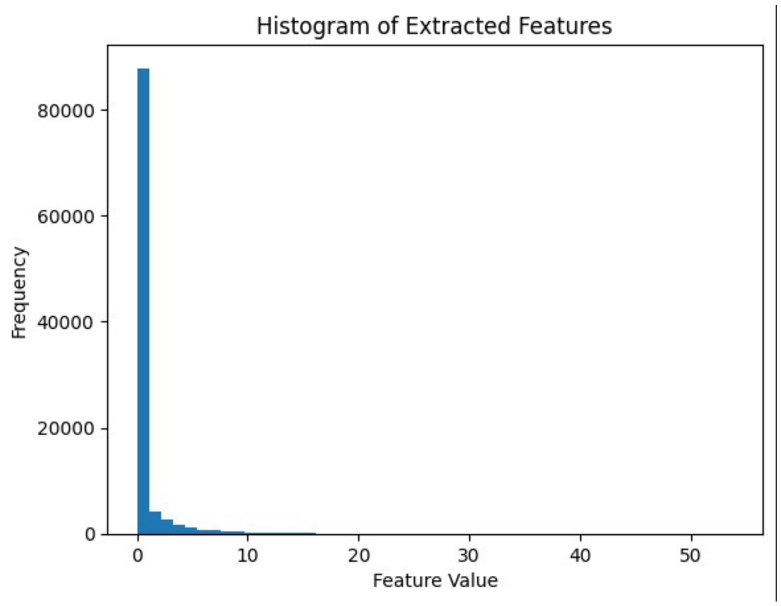
Code:

```
import numpy as np
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

# Load pre-trained ResNet50 model
resnet50_model = ResNet50(weights='imagenet', include_top=False)

# Define a new model with ResNet50 base
model = Model(inputs=resnet50_model.input,
outputs=resnet50_model.get_layer('conv5_block3_out').output)

# Function to extract features from images using ResNet50
def extract_features(img_path):
img = image.load_img(img_path, target_size=(224, 224)) # ResNet50 expects images of size 224x224
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x) # Preprocess the input data to align with the way ResNet50 was trained
features = model.predict(x)
```

```
return features.flatten() # Flatten the features to use them as input to another classifier

# Example usage
img_path = '/content/drive/MyDrive/cat.png'
features = extract_features(img_path)
print("Shape of extracted features:", features.shape)


# Function to plot histogram of extracted features
def plot_features_histogram(features):
plt.hist(features, bins=50)
plt.title('Histogram of Extracted Features')
plt.xlabel('Feature Value')
plt.ylabel('Frequency')
plt.show()

# Example usage
plot_features_histogram(features)
```

Output:

```
1/1 [==============================] - 2s 2s/step Shape of extracted features:
(100352,)
```



III Implementation of Pre-trained CNN models using transfer learning for classification/object detections.

a) AlexNet

b) VGG-16

Code:

a)

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

# Set device (GPU if available, else CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Define transformation for data augmentation and normalization
transform = transforms.Compose([
transforms.Resize((224, 224)),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Load CIFAR-10 dataset
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Create data loaders
train_loader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)

# Load pre-trained AlexNet model
alexnet_model = models.alexnet(pretrained=True)

# Freeze parameters of pre-trained layers
for param in alexnet_model.parameters():
param.requires_grad = False

# Modify the last fully connected layer for CIFAR-10 classification (10 classes)
num_features = alexnet_model.classifier[6].in_features
alexnet_model.classifier[6] = nn.Linear(num_features, 10)
```

```
# Move model to device
alexnet_model = alexnet_model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(alexnet_model.parameters(), lr=0.001)

# Train the model
num_epochs = 5
for epoch in range(num_epochs):
running_loss = 0.0
for i, (inputs, labels) in enumerate(train_loader):
inputs = inputs.to(device)
labels = labels.to(device)

optimizer.zero_grad()

outputs = alexnet_model(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()

running_loss += loss.item()

if (i+1) % 100 == 0:
print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], Loss:
{running_loss/100}')
running_loss = 0.0

# Evaluate the model
alexnet_model.eval()
correct = 0
total = 0
with torch.no_grad():
for inputs, labels in test_loader:
inputs = inputs.to(device)
labels = labels.to(device)
outputs = alexnet_model(inputs)
_, predicted = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy of the network on the 10000 test images: {accuracy}%')
```

b)

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Load pre-trained VGG16 model (without top fully connected layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the convolutional layers
for layer in base_model.layers:
layer.trainable = False

# Add custom classification layers
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)

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

# Train the model
model.fit(x_train, y_train, epochs=50, batch_size=64, validation_data=(x_test, y_test))
```

Output:

a)

```
Epoch [5/5], Step [100/782], Loss: 0.5746017411351204

Epoch [5/5], Step [200/782], Loss: 0.5881411558389664

Epoch [5/5], Step [300/782], Loss: 0.6102184489369392

Epoch [5/5], Step [400/782], Loss: 0.5950301320850849

Epoch [5/5], Step [500/782], Loss: 0.6262218597531318

Epoch [5/5], Step [600/782], Loss: 0.5968183997273445
Epoch [5/5], Step [700/782], Loss: 0.6252494990825653

Accuracy of the network on the 10000 test images: 82.13%
```

b)

```
Epoch 46/50

782/782 [==============================] - 11s 14ms/step - loss: 0.3186 -

accuracy: 0.8953 - val_loss: 1.7425 - val_accuracy: 0.5968

Epoch 47/50

782/782 [==============================] - 10s 13ms/step - loss: 0.3130 -

accuracy: 0.8953 - val_loss: 1.7340 - val_accuracy: 0.6060

Epoch 48/50

782/782 [==============================] - 10s 13ms/step - loss: 0.3026 -

accuracy: 0.8994 - val_loss: 1.7971 - val_accuracy: 0.5943

Epoch 49/50

782/782 [==============================] - 11s 14ms/step - loss: 0.2939 -

accuracy: 0.9028 - val_loss: 1.7883 - val_accuracy: 0.6012

Epoch 50/50

782/782 [==============================] - 11s 14ms/step - loss: 0.2841 -

accuracy: 0.9060 - val_loss: 1.8380 - val_accuracy: 0.6009
```

IV Practicing various strategies of fine tuning.

Code:

```
import tensorflow as tf

from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

```
# Freeze layers except the last convolutional block
for layer in base_model.layers[:-4]:
layer.trainable = False

# Add custom classification layers
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(lr=1e-4), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Print model summary
model.summary()

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))

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

Output:

```
Total params: 14848586 (56.64 MB)

Trainable params: 7213322 (27.52 MB)

Non-trainable params: 7635264 (29.13 MB)
```

```
Epoch 6/10

782/782 [==============================] - 16s 20ms/step - loss: 0.4759 -

accuracy: 0.8338 - val_loss: 0.8067 - val_accuracy: 0.7454

Epoch 7/10

782/782 [==============================] - 17s 22ms/step - loss: 0.4210 -

accuracy: 0.8527 - val_loss: 0.8570 - val_accuracy: 0.7322

Epoch 8/10

782/782 [==============================] - 16s 20ms/step - loss: 0.3660 -

accuracy: 0.8697 - val_loss: 0.9557 - val_accuracy: 0.7352
```

```
Epoch 9/10
782/782 [==============================] - 16s 21ms/step - loss: 0.3181 -
accuracy: 0.8870 - val_loss: 0.9987 - val_accuracy: 0.7391
Epoch 10/10
782/782 [==============================] - 16s 20ms/step - loss: 0.2770 -
accuracy: 0.9025 - val_loss: 1.0109 - val_accuracy: 0.7410 313/313
[==============================] - 3s 9ms/step - loss: 1.0109 - accuracy:
0.7410
```

V ImplementingGenerative Models:

a) Autoencoder for image reconstruction

b) Word Prediction using RNN

c) Image Captioning

Code:

a)

```
import numpy as np

import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist

# Load MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

# Define the autoencoder model
input_img = Input(shape=(28, 28, 1))
x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
```

```
encoded = MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

# Create autoencoder model
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the model
autoencoder.fit(x_train, x_train,
epochs=10,
batch_size=128,
shuffle=True,
validation_data=(x_test, x_test))

# Generate reconstructed images
decoded_imgs = autoencoder.predict(x_test)

# Plot some of the reconstructed images
n = 10 # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
# Display 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)

# Display 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()
```

b)

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import random
import string
import re
from sklearn.model_selection import train_test_split

# Sample list of sentences
text_data = [
"I like to eat apples and oranges",
"Apples and oranges are delicious fruits",
"I prefer oranges over apples",
"Oranges are juicy and delicious"
]

# Preprocess and tokenize text data
def preprocess_text(text):
text = text.lower()
text = re.sub(r'\d+', '', text) # Remove numbers
text = text.translate(str.maketrans('', '', string.punctuation)) # Remove punctuation
return text.split()

# Preprocess and tokenize text data
tokens = []
for sentence in text_data:
tokens.extend(preprocess_text(sentence))

# Create sequences of input and target pairs
seq_length = 5
sequences = []
for i in range(len(tokens) - seq_length):
sequences.append((tokens[i:i+seq_length], tokens[i+seq_length]))

# Create word-to-index and index-to-word mappings
word_to_idx = {word: idx for idx, word in enumerate(set(tokens))}
idx_to_word = {idx: word for word, idx in word_to_idx.items()}
vocab_size = len(word_to_idx)

# Define a PyTorch Dataset
class TextDataset(Dataset):
def __init__(self, sequences):
self.sequences = sequences
```

```
def __len__(self):
return len(self.sequences)

def __getitem__(self, idx):
input_seq, target = self.sequences[idx]
input_idx = torch.tensor([word_to_idx[word] for word in input_seq], dtype=torch.long)
target_idx = torch.tensor(word_to_idx[target], dtype=torch.long)
return input_idx, target_idx

# Split data into train and test sets
train_data, test_data = train_test_split(sequences, test_size=0.2, random_state=42)

# Create DataLoader instances
train_loader = DataLoader(TextDataset(train_data), batch_size=64, shuffle=True)
test_loader = DataLoader(TextDataset(test_data), batch_size=64, shuffle=False)

# Define the LSTM model
class WordPredictionModel(nn.Module):
def __init__(self, vocab_size, embedding_dim, hidden_dim):
super(WordPredictionModel, self).__init__()
self.embedding = nn.Embedding(vocab_size, embedding_dim)
self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
self.fc = nn.Linear(hidden_dim, vocab_size)

def forward(self, x):
embedded = self.embedding(x)
lstm_out, _ = self.lstm(embedded)
out = self.fc(lstm_out[:, -1, :])
return out

# Initialize the model, loss function, and optimizer
embedding_dim = 100
hidden_dim = 128
model = WordPredictionModel(vocab_size, embedding_dim, hidden_dim)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
num_epochs = 10
for epoch in range(num_epochs):
running_loss = 0.0
for input_idx, target_idx in train_loader:
optimizer.zero_grad()
output = model(input_idx)
loss = criterion(output, target_idx)
loss.backward()
optimizer.step()
```

```
running_loss += loss.item()
print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss / len(train_loader):.4f}')

# Function to generate next words
def generate_next_words(seed_text, next_words):
with torch.no_grad():
seed_seq = seed_text.split()
for _ in range(next_words):
input_idx = torch.tensor([word_to_idx[word] for word in seed_seq], dtype=torch.long).unsqueeze(0)
output = model(input_idx)
predicted_idx = torch.argmax(output, dim=1).item()
next_word = idx_to_word.get(predicted_idx, '<UNK>')
seed_seq.append(next_word)
return ' '.join(seed_seq)

# Generate and print next words
seed_text = "apples are delicious"
print(generate_next_words(seed_text, 5))
```

c)

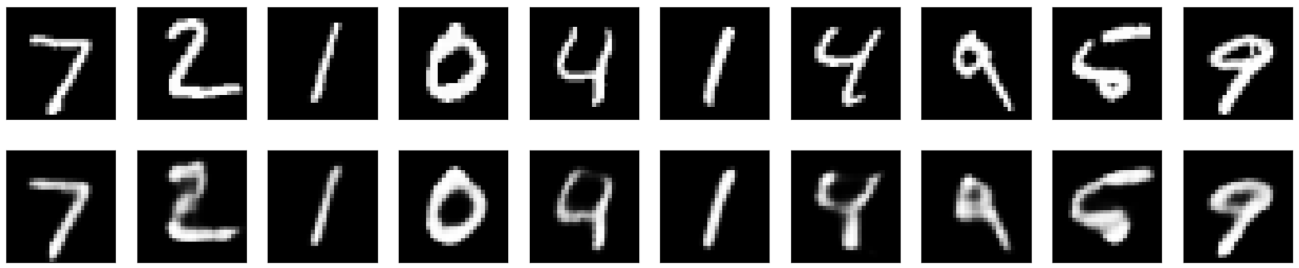
```
import matplotlib.pyplot as plt


import torch
from torchvision.transforms import transforms
from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration
import nltk
nltk.download('punkt')

# Load the pre-trained image captioning model
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

# Load and preprocess the image
image_path = "/content/cat.jfif"
image = Image.open(image_path).convert("RGB") # Convert to RGB
preprocess = transforms.Compose([
transforms.Resize((256, 256)),
transforms.ToTensor(),
])
input_tensor = preprocess(image).unsqueeze(0)

# Generate captions
with torch.no_grad():
```

```
captions = model.generate(pixel_values=input_tensor)

# Decode the generated captions
caption_text = processor.decode(captions[0], skip_special_tokens=True)

# Print the generated caption
print("Generated Caption:", caption_text)

plt.imshow(image)
plt.axis('off')
plt.show
```

Output:

a)

```
Epoch 6/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1126 -
val_loss: 0.1102
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1108 -
val_loss: 0.1085
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1091 -
val_loss: 0.1070
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1079 -
val_loss: 0.1058
Epoch 10/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1068 -
val_loss: 0.1052
313/313 [==============================] - 1s 2ms/step
```



b)

```
Epoch [6/10], Loss: 2.2166
Epoch [7/10], Loss: 2.1457
Epoch [8/10], Loss: 2.0729
Epoch [9/10], Loss: 1.9981
Epoch [10/10], Loss: 1.9212
apples are delicious fruits fruits prefer oranges apples
```

c)

```
Generated Caption: a group of cats sitting in a row
```