TASK-3

1. Comparison of CNN and Yolo
2. Try implementing CNN and Yolo with built-in functions for any dataset
3. down the results and accuracy
4. Different layers in CNN and your understanding

Convolutional Neural Network (CNN):

CNN is a deep learning algorithm widely used for image classification, object detection, and various computer vision tasks. It is designed to automatically learn hierarchical representations of data directly from pixel values. CNNs are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

CNN layers:

1. Input Layer: The first layer takes the raw image data as input.

2. Convolutional Layer: This layer applies a set of learnable filters to the input image, performing convolution operations. Each filter extracts different features from the image by sliding across the input and computing dot products.

3. Activation Layer: Usually placed after the convolutional layer, this layer applies a non-linear activation function (e.g., ReLU) element-wise to introduce non-linearity into the network.

4. Pooling Layer: Also known as the subsampling layer, it reduces the spatial dimensions of the input by downsampling. Common pooling techniques include max pooling and average pooling.

5. Fully Connected Layer: This layer connects every neuron from the previous layer to the subsequent layer, similar to traditional neural networks. It learns complex patterns by combining features extracted by earlier layers.

6. Output Layer: The final layer provides the desired output, such as class probabilities for image classification.

Code Implementation

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

# Define the CNN model
model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    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'])

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

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

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

YOLO (You Only Look Once):

YOLO is a real-time object detection algorithm that directly predicts bounding boxes and class probabilities in a single pass through the network. It is known for its speed and accuracy in object detection tasks. YOLO divides the input image into a grid and predicts bounding boxes and class probabilities within each grid cell.

YOLO components:

1. Input Processing: The input image is resized and processed to match the network's required input size.

2. Backbone Network: YOLO typically uses a pre-trained CNN, such as Darknet or MobileNet, as the backbone network. This network extracts features from the input image.

3. Detection Layers: These layers are responsible for predicting bounding boxes and class probabilities. YOLO uses anchor boxes, which are pre-defined boxes of various sizes and aspect ratios, to make predictions.

4. Non-max Suppression: YOLO applies non-max suppression to remove redundant bounding boxes and keep only the most confident ones.

Implementation

```
import cv2
import numpy as np
from darknet import darknet

# Load YOLO pre-trained weights and configuration
net, class_names, _ = darknet.load_network("yolov4.cfg", "yolov4.weights", "coco.names")

# Define the input image
image_path = "image.jpg"

# Load the image
image = cv2.imread(image_path)

# Resize the image to match the network's input size
resized_image = cv2.resize(image, (darknet.network_width(net), darknet.network_height(net)),
interpolation=cv2.INTER_LINEAR)

# Create a blob from the resized image
blob = darknet.blob_from_image(resized_image, 1 / 255.0, (darknet.network_width(net),
darknet.network_height(net)), swapRB=True, crop=False)

# Set the network input
darknet.network_input(net, blob)

# Perform object detection
detections = darknet.detect_image(net, class_names, image)

# Print the detections
for detection in detections:
    class_name, confidence, (x, y, w, h) = detection
    print(f'Class: {class_name}, Confidence: {confidence}, Bounding Box: {x}, {y}, {w}, {h}')

# Display the image with bounding boxes
image = darknet.draw_boxes(detections, image, class_names)

cv2.imshow("Object Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```