**Object Detection and Classification**

**1. Convolutional Neural Networks (CNN)**

**Theory:**  
CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images using layers like convolution, pooling, and fully connected layers.

**Applications:**

* Image classification
* Object detection
* Medical imaging
* Facial recognition

**Python Code:**

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

# 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 # Normalize pixel values

# 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.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

# Compile and train the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=3, validation\_data=(x\_test, y\_test))

# Predict on a test image

test\_image = x\_test[0]

plt.imshow(test\_image)

plt.title("Test Image")

plt.show()

prediction = model.predict(np.expand\_dims(test\_image, axis=0))

print("Predicted class:", np.argmax(prediction))

**2. Region-based CNN (R-CNN)**

**Theory:**  
R-CNN first uses selective search to propose regions, then extracts features using a CNN for each region and classifies them with SVM.

**Applications:**

* Object detection
* Image segmentation
* Localization

**Conceptual Code**

import cv2

import numpy as np

import torch

import torchvision.models as models

import torchvision.transforms as transforms

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification\_report

from tqdm import tqdm

import os

# 1. Load image and generate region proposals

def get\_proposals(image):

ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

ss.setBaseImage(image)

ss.switchToSelectiveSearchFast()

rects = ss.process()

return rects[:200] # top 200 proposals

# 2. Feature extractor using VGG16 (like original R-CNN)

vgg = models.vgg16(pretrained=True).features.eval()

transform = transforms.Compose([

transforms.ToPILImage(),

transforms.Resize((224, 224)),

transforms.ToTensor()

])

def extract\_feature(region):

with torch.no\_grad():

tensor = transform(region).unsqueeze(0)

features = vgg(tensor)

return features.view(-1).numpy()

# 3. Simulate labeled data (for training purposes only)

def create\_dataset(image\_path, label):

image = cv2.imread(image\_path)

proposals = get\_proposals(image)

features, labels = [], []

for (x, y, w, h) in proposals:

roi = image[y:y+h, x:x+w]

if roi.shape[0] > 0 and roi.shape[1] > 0:

try:

feat = extract\_feature(roi)

features.append(feat)

labels.append(label)

except:

continue

return features, labels

# 4. Train SVM on extracted features

positive\_feats, positive\_labels = create\_dataset("cat.jpg", "cat")

negative\_feats, negative\_labels = create\_dataset("dog.jpg", "dog")

X = np.array(positive\_feats + negative\_feats)

y = np.array(positive\_labels + negative\_labels)

y\_encoded = LabelEncoder().fit\_transform(y)

print("Training SVM...")

clf = SVC(kernel='linear')

clf.fit(X, y\_encoded)

# 5. Test on a new image

def test\_rcnn(image\_path):

image = cv2.imread(image\_path)

proposals = get\_proposals(image)

for (x, y, w, h) in proposals[:100]:

roi = image[y:y+h, x:x+w]

if roi.shape[0] > 0 and roi.shape[1] > 0:

try:

feat = extract\_feature(roi)

pred = clf.predict([feat])

label = LabelEncoder().inverse\_transform(pred)

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.putText(image, label[0], (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)

except:

continue

cv2.imshow("RCNN Detection", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Run detection

test\_rcnn("test.jpg")

**3. Fast R-CNN**

**Theory:**  
Improves R-CNN by feeding the entire image into a CNN once and extracting features. Region proposals are pooled and classified using ROI Pooling.

**Applications:**

* Real-time object detection
* Autonomous vehicles

**Conceptual Code:**

!pip install torch torchvision matplotlib opencv-python

import torch

from torchvision.models.detection import fasterrcnn\_resnet50\_fpn

from torchvision.transforms import functional as F

from PIL import Image

import matplotlib.pyplot as plt

import cv2

# Load pre-trained Faster R-CNN (which includes Fast R-CNN classifier)

model = fasterrcnn\_resnet50\_fpn(pretrained=True)

model.eval()

# Load and preprocess test image

image\_path = "test.jpg" # Replace with your image

image = Image.open(image\_path).convert("RGB")

image\_tensor = F.to\_tensor(image).unsqueeze(0) # Add batch dimension

# Run inference

with torch.no\_grad():

predictions = model(image\_tensor)

# Visualize results

img\_cv = cv2.imread(image\_path)

for idx in range(len(predictions[0]['boxes'])):

box = predictions[0]['boxes'][idx].cpu().numpy().astype(int)

score = predictions[0]['scores'][idx].item()

label = predictions[0]['labels'][idx].item()

if score > 0.5: # confidence threshold

cv2.rectangle(img\_cv, (box[0], box[1]), (box[2], box[3]), (0, 255, 0), 2)

cv2.putText(img\_cv, f"Class: {label}, Score: {score:.2f}",

(box[0], box[1]-10), cv2.FONT\_HERSHEY\_SIMPLEX,

0.5, (0, 255, 0), 1)

cv2.imshow("Fast R-CNN Predictions", img\_cv)

cv2.waitKey(0)

cv2.destroyAllWindows()

**4. YOLO (You Only Look Once)**

**Theory:**  
YOLO divides the input image into an SxS grid. Each grid cell predicts bounding boxes and class probabilities, making it extremely fast.

**Applications:**

* Real-time detection in videos
* Surveillance systems
* Autonomous driving

**Python Code:**

import cv2

import numpy as np

# Load the pre-trained YOLOv3 model

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Load and prepare test image

image = cv2.imread("test.jpg")

height, width = image.shape[:2]

blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)

# Perform detection

net.setInput(blob)

outputs = net.forward(output\_layers)

# Draw bounding boxes

for output in outputs:

for detection in output:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x, center\_y, w, h = (detection[0:4] \* np.array([width, height, width, height])).astype('int')

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow("YOLO Detection", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**5. Vision Transformer (ViT)**

**Theory:**  
ViT treats image patches as tokens and applies Transformer encoders to capture global image features, avoiding convolution layers altogether.

**Applications:**

* Image classification
* Medical image analysis
* Fine-grained recognition tasks

**Python Code:**

from transformers import ViTFeatureExtractor, ViTForImageClassification

from PIL import Image

import torch

# Load image

image = Image.open("test.jpg").convert("RGB")

# Load model and feature extractor

feature\_extractor = ViTFeatureExtractor.from\_pretrained("google/vit-base-patch16-224")

model = ViTForImageClassification.from\_pretrained("google/vit-base-patch16-224")

# Preprocess and predict

inputs = feature\_extractor(images=image, return\_tensors="pt")

outputs = model(\*\*inputs)

logits = outputs.logits

predicted\_class = logits.argmax(-1).item()

print("Predicted Class ID:", predicted\_class)