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)
```

```python
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:

```python
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)
```