Introduction

Object recognition is a key challenge in the field of computer vision, and it has a wide range of applications, including robotics, self-driving cars, surveillance, and augmented reality. Over the years, two main approaches have emerged to address this challenge. The first is traditional computer vision (CV), which relies on handcrafted features and classical machine learning algorithms. The second is deep learning, which utilizes convolutional neural networks (CNNs) to learn features directly from data.

In this coursework, we'll be comparing the performance of these two approaches in object recognition tasks. We'll be using two distinct datasets for our analysis: CIFAR-10 and the RGB-D Object Dataset. CIFAR-10 is a well-known benchmark that includes 60,000 labelled RGB images across 10 different object categories, making it a great resource for image classification. On the other hand, the RGB-D Object Dataset presents a more intricate challenge, containing over 300 object instances across 51 categories, with RGB-D frames captured in real-world environments.

The goal of our comparison is to explore the trade-offs between traditional computer vision methods and deep learning techniques. We'll look at various factors, including accuracy, computational efficiency, interpretability, and data requirements. Traditional methods, such as the Bag of Visual Words (BoVW) combined with Support Vector Machines (SVMs), tend to be lightweight and interpretable. However, they often struggle to handle complex visual variations. Deep learning models, particularly CNNs, usually deliver better performance but come with the need for large datasets, extensive computational resources, and precise tuning.

By conducting experiments with both datasets and examining both methodologies, this report aims to offer a thoughtful and balanced analysis of the practical considerations and performance outcomes of traditional and modern approaches to object recognition.

Methodology

Datasets

We evaluated both traditional and deep learning approaches on two object recognition datasets with differing characteristics: the widely used **CIFAR-10** (taken from Kaggle) and the more realistic **RGB-D Object Dataset (Cropped Evaluation Set)**. These datasets were chosen to assess how each method generalizes across synthetic and real-world visual settings.

CIFAR-10

- **Description**: A standard benchmark in image classification, CIFAR-10 contains 60,000 RGB images of size 32×32 across 10 categories, including animals (e.g., dog, cat) and vehicles (e.g., airplane, car).
- **Split**: The dataset is pre-divided into 50,000 training and 10,000 test images. For our pipeline, 20% of the training set was reserved for validation.
- **Preprocessing**:
    - Images were resized to 224×224 to improve key point detection for SIFT and to comply with CNN input dimensions.

- For CNNs, data augmentations such as random horizontal flips, normalization, and cropping were applied.

RGB-D Object Dataset (Cropped Evaluation Set)

- **Description**: This dataset contains 51 object categories, each represented by several object instances (e.g., apple_1, mug_2, soda_bottle_3). Each instance is recorded from three video sequences and subsampled every 5th frame, resulting in a smaller and more computationally feasible subset.
- **Cropped Format**: The images are tightly cropped around the object, as used in the original object recognition benchmark.

Leakage Risk: A key challenge is that multiple frames per instance are highly correlated. Randomly splitting images can lead to data leakage. To prevent data leakage and ensure a fair evaluation of generalization, we avoided random per-image splitting and instead performed instance-based splitting. This method prevents overestimation of model performance due to memorization of instance-specific visual features. Since multiple images in the dataset represent the same object instance (e.g., apple_1_1.png, apple_1_2.png), splitting them across training and test sets would allow the model to see nearly identical views of the same object during training and evaluation

Traditional Computer Vision Approach (SIFT + BoVW + SVM)

Feature Extraction using SIFT

We employed SIFT (Scale-Invariant Feature Transform) to extract local features from grayscale images. SIFT was chosen for its robustness to changes in scale, rotation, and illumination — making it well-suited for both controlled datasets like CIFAR-10 and real-world datasets like RGB-D. Before feature extraction, all images were resized to 224×224 to ensure sufficient key point coverage.

```python
def extract_sift_features(image_paths, max_features=500):
    sift = cv2.SIFT_create(nfeatures=max_features)
    descriptors = []
    valid_paths = []

    for path in image_paths:
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        if img is None:
            continue
        kp, des = sift.detectAndCompute(img, None)
        if des is not None:
            descriptors.append(des)
            valid_paths.append(path)

    return descriptors, valid_paths
```

Bag-of-Visual-Words (BoVW) Representation

The core idea behind BoVW is to transform local SIFT descriptors into a fixed-length global feature vector for each image:

- **Descriptor Aggregation**: For each training image, SIFT descriptors were extracted. These descriptors from all training images were pooled into a single array.
- **Vocabulary Construction**: The aggregated descriptors were clustered using MiniBatch K-Means with a vocabulary size of 100 visual words as a trade-off between representation power and training time.

```python
print("Clustering descriptors to build BoVW vocabulary...")
kmeans = MiniBatchKMeans(n_clusters=num_clusters, batch_size=1000, verbose=1)
kmeans.fit(descriptor_stack)

# --- Step 4: Compute BoVW histograms for training images ---
def compute_bovw_histograms(images):
    histograms = []
    for img in images:
        keypoints, descriptors = sift.detectAndCompute(img, None)
        if descriptors is not None:
            words = kmeans.predict(descriptors)
            hist, _ = np.histogram(words, bins=np.arange(num_clusters+1), density=True)
        else:
            hist = np.zeros(num_clusters)  # no keypoints
        histograms.append(hist)
    return np.array(histograms)

print("Computing BoVW histograms for training data...")
train_bovw = compute_bovw_histograms(train_images)
```

Larger vocabularies (e.g., 200+) risk overfitting on smaller class subsets and significantly increase feature dimensionality.

- **Histogram Encoding**: Each image's descriptors were then assigned to the nearest cluster centre (visual word), forming a **histogram over the vocabulary**. This histogram was L2-normalized and used as the image representation.

```python
def build_vocabulary(descriptor_list, k=100):
    descriptors = np.vstack(descriptor_list).astype(np.float32)
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(descriptors)
    return kmeans


def extract_bow_features(kmeans, descriptor_list, k=100):
    features = []
    for descriptors in descriptor_list:
        if descriptors is None or len(descriptors) == 0:
            features.append(np.zeros(k))
            continue
        descriptors = descriptors.astype(np.float32)
        words = kmeans.predict(descriptors)
        hist = np.zeros(k)
        for w in words:
            hist[w] += 1
        features.append(hist)
    return np.array(features, dtype=np.float32)
```

Classification with SVM

The BoVW histograms were standardized using **StandardScaler**, and a **linear Support Vector Machine (SVM)** was trained to perform multi-class classification. This classifier was trained separately on both datasets:

- On **CIFAR-10**, we used the class-wise folders and trained/test split provided in the dataset.
- For **RGB-D**, instance-level splitting was used to prevent leakage, and each image's BoVW features were mapped to its class label.

Evaluation and Visualization

Model performance was evaluated using:

- **Accuracy** and **classification reports** (precision, recall, F1-score).
- **Confusion matrices** to visualize class-wise performance.
- **Per-class precision bar plots** to identify which categories were learned well versus poorly.

MiniBatch KMeans was used to scale clustering across tens of thousands of SIFT descriptors.

The **SVM classifier** used a **linear kernel** to avoid unnecessary complexity and because preliminary tests with non-linear kernels (like RBF) offered minimal gains at much higher cost. Standard scaling was applied to the BoVW histograms before training to normalize feature ranges. No formal grid search or cross-validation was applied due to computational constraints.

CNN-Based Deep Learning Approach

CIFAR-10: Custom CNN from Scratch

Since CIFAR-10 is a standard benchmark with ample training data (50,000 training images), a small CNN was trained from scratch. The architecture typically consisted of:

3 convolutional layers with ReLU activations and MaxPooling, followed by a flattening layer and two fully connected layers, using Softmax for classification over 10 classes. The learning rate was set to **1e-4** as a stable starting point for both training from scratch (CIFAR-10) and fine-tuning (RGB-D). A smaller rate was preferred for ResNet-18 to avoid overwriting pretrained weights too aggressively.

Adam was chosen for its adaptive learning capabilities and ease of tuning. A **batch size of 32** balanced convergence speed with memory constraints on GPU.

Training: ~10–20 was selected based on early training curves; overfitting began to emerge in the RGB-D dataset after ~10–15 epochs due to limited variation in some categories. **Early stopping** was excluded in the final version to capture full learning curves, but validation metrics were monitored manually. Dropout and weight decay were not extensively tuned, as the primary focus was on comparing overall method performance.

RGB-D Dataset: Transfer Learning with ResNet-18

Given the smaller and more variable RGB-D evaluation dataset (subsampled instances), transfer learning with ResNet-18 was employed. This leverages robust features learned from ImageNet to overcome the limited size of the dataset.

- The final FC layer of ResNet was replaced to match the number of classes (51).
- The base layers were optionally frozen or fine-tuned.

Training Strategy:

- Use of learning rate scheduling
- Weighted sampling or data augmentation if needed to balance classes

---

CIFAR-10 Results

Traditional Method: SIFT + BoVW + SVM

The traditional pipeline using SIFT-based Bag-of-Visual-Words and a linear SVM achieved modest results:

- **Test Accuracy**: **28.71%**
- **Macro-Averaged F1-Score**: **0.29**

Despite leveraging handcrafted descriptors, the performance was limited due to:

- CIFAR-10's **low resolution (32×32)**, which restricts the effectiveness of keypoint-based descriptors like SIFT.
- **Loss of spatial context** in BoVW histograms, making it hard to distinguish between visually similar classes.
- The confusion matrix showed widespread misclassifications, particularly between animal categories (e.g., cat, dog, horse), reflecting a lack of semantic richness in the feature representation.

Test Accuracy: 28.71%

	precision	recall	f1-score	support
airplane	0.34	0.37	0.36	1000
automobile	0.34	0.37	0.35	1000
bird	0.23	0.18	0.20	1000
cat	0.18	0.12	0.15	1000
deer	0.21	0.18	0.19	1000
dog	0.27	0.30	0.28	1000
frog	0.26	0.33	0.29	1000
horse	0.30	0.27	0.28	1000
ship	0.34	0.37	0.35	1000
truck	0.33	0.40	0.36	1000
accuracy			0.29	10000
macro avg	0.28	0.29	0.28	10000
weighted avg	0.28	0.29	0.28	10000

Deep Learning Method: CNN (Custom Architecture)

A custom-built CNN trained from scratch significantly outperformed the traditional pipeline:

- **Test Accuracy**: **78.84%**
- **Macro-Averaged F1-Score**: **0.79**
- **Per-Class Performance**:
    - Highest precision: **Ship (0.92)**, **Automobile (0.90)**
    - Lowest precision: **Cat (0.60)**, **Bird (0.70)**

The CNN's ability to learn hierarchical spatial features allowed it to better distinguish between object categories, even in a low-resolution setting. The confusion matrix shows strong diagonal dominance, indicating that the model correctly classifies most test examples.

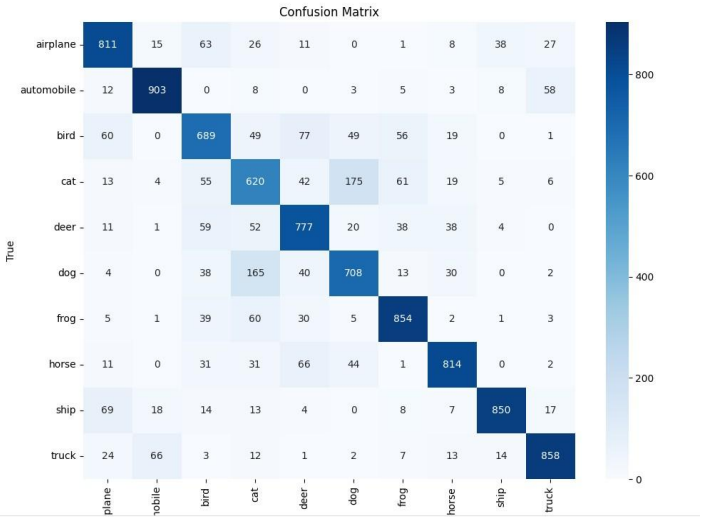
```
Test Accuracy: 0.7884

Classification Report:
              precision    recall  f1-score   support

    airplane       0.80      0.81      0.80      1000
  automobile       0.90      0.90      0.90      1000
        bird       0.70      0.69      0.69      1000
         cat       0.60      0.62      0.61      1000
        deer       0.74      0.78      0.76      1000
         dog       0.70      0.71      0.71      1000
        frog       0.82      0.85      0.84      1000
       horse       0.85      0.81      0.83      1000
        ship       0.92      0.85      0.89      1000
       truck       0.88      0.86      0.87      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
weighted avg       0.79      0.79      0.79     10000
```

RGB-D Results

Deep Learning Method: ResNet-18 (Transfer Learning)

Using a pretrained ResNet-18 model fine-tuned on the 51-class RGB-D cropped evaluation dataset produced strong performance:

- **Test Accuracy**: **72.36%**
- **Macro-Averaged F1-Score**: **0.68**
- **Weighted-Averaged F1-Score**: **0.72**
- **Total Images Evaluated**: 31,696 RGB frames



Observations

- Most classes achieved high performance (e.g., *food_cup*, *toothpaste*, *sponge*), with precision and recall >0.90.
- A few classes like *marker*, *mushroom*, and *comb* had F1-scores below 0.25 due to low recall — possibly caused by class imbalance or visual similarity with other categories.
- The **confusion matrix** shows strong diagonal dominance, indicating accurate classification across most classes.

Traditional Method: SIFT + BoVW + SVM: The traditional approach, using SIFT descriptors with Bag-of-Visual-Words (BoVW) and a linear Support Vector Machine, was applied to the RGB-D Cropped Eval dataset with 51 object categories. The vocabulary size was set to 100 clusters.

Performance Summary

- **Validation Accuracy**: **34.25%, Test Accuracy**: **34.95%, Macro F1-Score (Test)**: **0.30, Weighted F1-Score (Test)**: **0.34**

Observations

- While the overall accuracy (~35%) is **substantially better than chance**, performance across classes was **highly uneven**.
- Some classes such as **marker (F1 = 0.56)**, **pliers (F1 = 0.57)**, and **glue_stick (F1 = 0.58)** performed well, likely due to distinctive shapes and textures.
- However, many classes had **F1-scores near 0.10–0.20**, with a few (e.g., *mushroom*, *peach*, *pitcher*) receiving **almost no correct predictions**.
- The **macro-averaged F1-score of 0.30** suggests that the model did not generalize well across all classes, especially for minority or visually ambiguous categories.
- These issues stem from the **limitations of BoVW**, such as its disregard for spatial information and its dependence on reliable local features, which can vary significantly across RGB-D frames.

Comparison with CNN

- While the CNN model (ResNet-18) achieved a **test accuracy of ~72%**, the traditional method plateaued at **~35%**.
- The CNN showed **much more consistent class-wise performance**, with several classes exceeding 90% F1-score.
- The deep model could learn **semantic and spatial hierarchies**, while BoVW relied only on local patch appearances, making it vulnerable to pose, scale, and background variation.

State of the Art in Computer Vision for Robotics

Deep learning has dramatically reshaped the landscape of robotic vision, fundamentally altering how robots perceive and engage with their surroundings. Traditional methods for computer vision, which relied on techniques like edge detection, feature matching, and geometric reasoning, are increasingly being supplanted or enhanced by deep neural networks that can be trained end-to-end. These modern approaches provide superior robustness, adaptability, and overall performance.

Key Trends in Robotic Vision

In modern robotics, **deep learning models dominate object recognition, segmentation, detection, and scene understanding**:

- **Object detection** has been revolutionized by models like **YOLOv7** [Wang et al., 2023], which balance real-time speed with high accuracy, and **Faster R-CNN** [Ren et al., 2015], known for its two-stage detection pipeline and region proposal mechanism.

- **Semantic and instance segmentation** with networks like **Mask R-CNN** and **DeepLab** enables pixel-wise classification, critical for grasp planning, obstacle avoidance, and manipulation.
- **3D object recognition** leverages RGB-D data and LiDAR through architectures like **PointNet** and **PointNet++** [Qi et al., 2017], which operate directly on point clouds and are widely used in autonomous robotics.
- **Transformers** (e.g., **DETR** [Carion et al., 2020] and **Swin Transformer** [Liu et al., 2021]) are gaining traction for their ability to model long-range dependencies in visual data, although they are still computationally heavy for many robotic platforms.

The Role of Deep Learning in Robotics

- **End-to-end learning** from raw sensory inputs to task-specific outputs.
- **Transfer learning**, where pretrained models such as ResNet or MobileNet are fine-tuned for robotic domains with limited data.
- **Improved generalization** under variable lighting, viewpoints, occlusions, and clutter.
- **Real-time performance**, especially when using optimized lightweight models like **YOLO-Nano** or **MobileNetV2** on embedded systems.

Current Limitations

Despite its strengths, deep learning in robotics faces several challenges:

- **Heavy data requirements**, which are impractical in dynamic or safety-critical robotic deployments.
- **Lack of interpretability**, especially when compared to traditional handcrafted pipelines such as SIFT or BoVW.
- **Hardware constraints** in edge robotics, where real-time inference must happen on devices like microcontrollers or Jetson boards.

Hybrid and Emerging Approaches

New research explores **hybrid models** that combine handcrafted features (e.g., keypoints, geometric descriptors) with CNNs, yielding robust SLAM and object tracking systems. Additionally, **self-supervised learning** methods like SimCLR and BYOL are being investigated to reduce the reliance on annotated data. The growing field of **Edge AI** enables deploying CNNs on low-power devices via pruning, quantization, and model compression.

References

- Ren, S. et al. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. NeurIPS.
- Wang, C.-Y. et al. (2023). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art*. arXiv preprint.

- Qi, C.R. et al. (2017). *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. CVPR.
- Carion, N. et al. (2020). *End-to-End Object Detection with Transformers*. ECCV.
- Liu, Z. et al. (2021). *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. ICCV.

Appendix

Code Repositories and Notebooks

- **CIFAR-10 BoVW + SVM**: https://www.kaggle.com/code/mananmal/cifar-10-traditional
- **CIFAR-10 CNN (Custom)**: https://www.kaggle.com/code/mananmal/cifar-10-cnn
- **RGB-D BoVW + SVM**: https://www.kaggle.com/code/mananmal/rgb-d-traditional
- **RGB-D CNN (ResNet-18)**: https://www.kaggle.com/code/mananmal/cnn-for-rgb-d