Comparison of CNN and Traditional Computer Vision

Anonymous Author(s)
Affiliation
Address
`email`

1 Preprocessing

1.1 Datasets

Two datasets were used: The STL-10[1] dataset provides 96×96 colour images. It contains $5\,000$ labelled training images (500 per class and 10 classes) and $8\,000$ test images. An additional $100\,000$ unlabelled images are available for unsupervised representation learning but are not used in our supervised experiments. For the Mammals dataset (Kaggle "45MammalSpecies" challenge) 20 species are selected, each represented by about 300 colour images at 256×256 resolution, totalling approximately $6,000$ images. Because no official split is provided, we randomly partition the data into training and test sets with an $80:20$ ratio, stratified by class.

1.2 Normalisation

Pre-processing is performed in two stages. First, every image is resized to a common spatial resolution of 96×96 pixels using OpenCV's bilinear interpolation, which assigns each output pixel the weighted average of its four nearest neighbours in the source image. The image is then converted to grayscale, yielding arrays of shape $(1, 96, 96)$. In the second stage the intensity values x are contrast-normalised using histogram equalisation or minmax linear scaling. Both procedures enhance contrast, the former by a linear stretch and the latter by a non-linear redistribution of intensities.

1.3 Feature detection

Local feature detection is examined under three settings: SIFT keypoints, a dense grid, and the absence of keypoints (raw images). **SIFT** (Scale-Invariant Feature Transform) [2] constructs a scale-space pyramid by convolving the image with Gaussians of increasing standard deviation σ. Differences of successive Gaussian-blurred images (DoG) are computed and local extrema are detected by comparing each pixel with its 26 neighbours across scale and space. Keypoints with weak contrast or lying on edges are discarded by thresholding the DoG magnitude and the principal curvature ratio derived from the Hessian matrix. For the **dense** configuration, keypoints are sampled every fifth pixel on a regular grid at the base image resolution, ensuring uniform coverage. In the **none** configuration no keypoint detection is performed; instead, the complete image is passed downstream.

1.4 Descriptors

Once keypoints are defined, local image regions are encoded using either SIFT descriptors or a quantized histogram of local pixel intensities. **SIFT**: For each keypoint a 16×16 neighbourhood in the appropriate scale is divided into sixteen 4×4 sub-regions. For every sub-region the gradient orientations are accumulated into an 8-bin histogram, producing a $4 \times 4 \times 8 = 128$-dimensional vector, which is subsequently ℓ_2-normalised to unit length. The key advantage of SIFT is that it generates rotationally robust descriptors of images that are robust to moderate changes in lighting as they use gradient rather than absolute values. In comparison histograms of local pixel intensities do not provide orientation information, and are not robust to illumination changes, they are used here as a comparison to SIFT to examine the value of rotationally invariant descriptors for class level classification.

1.5 Bag of Words

In the Bag of Words (BoW) [3] approach local feature descriptors extracted from training images are first clustered; each cluster centre is assigned a unique index, and an input image is represented by a histogram that counts how many of its descriptors are assigned to each cluster. The number of clusters k therefore defines the dimensionality of the feature space that is subsequently passed to a classifier. In our implementation descriptors are clustered with K–means.

1.5.1 *k*-means clustering

k-means is an iterative partitional clustering algorithm that seeks a set of k centroids $\mu_1, \ldots, \mu_k$ that minimise the within–cluster sum of squared Euclidean distances where $\mathbf{x}_i$ is the i–th descriptor and $c(i) \in 1, \ldots, k$ is the index of the centroid assigned to $\mathbf{x}_i$. There are alternative and more expressive methods for clustering such as gaussian mixture models or probabilistic PCA, however K-means is more computationally efficient that these algorithms.

1.6 Classification

Given either the Bag of Words histogram or the raw vectorised pixel values, two classifiers are evaluated: a Random Forest (RF) ensemble and the *k*-nearest neighbours (*k*NN) algorithm. The **Random Forest** classifier is an ensemble of B decision trees grown on independent bootstrap samples of the training data. Random Forests are computationally fast to train and powerful models for learning non-linear decision boundaries, making them great models for quickly iterating on and improving cV pipelines.. The ***k*NN** classifier classifies query points $\mathbf{x}_*$ by computing the distance (Minkowski metric with parameter p) of the point to all training samples, identifies the k closest instances, and assigns the label that is most common (*uniform* weighting) or that achieves the highest sum of inverse-distance weights (*distance* weighting) among those neighbours. The only learnable parameter is k and model performance is deterministic and non-parametric, KNNs are fast to train but fail under certain assumptions of distance making them an effective baseline comparison for metric learners like Random Forests.

1.7 CNN for vision

Convolutional Neural Networks (CNNs) stack layers of learnable kernels to generate intermediate image representations. Intermediate image representations can be stacked by increasing the kernel depth, which allows for exploitation of not only spatially relevant information but also of channel specific information. By appending a classification head to the final representational output of the CNN, the network can be trained end to end to discriminate images by learning both feature specific kernels and decision boundaries automatically.

1.7.1 CNN Parameters

For this work a classic small CNN is used which accepts 96×96 resolution images with either 1 or 3 colour channels, and has a classification head composed of linear layer with an output size of 10. In addition to the baseline 96 × 96 classifier, a lightweight convolutional network that builds multi-scale awareness directly into each block is used. Inspired by the scale-space pooling of SIFT descriptors, every MultiKernelBlock processes the same feature map in parallel with 3×3, 5×5, and 7×7 kernels, then concatenates the resulting activations. This channel-splitting strategy allows the model to capture fine texture and broad structure simultaneously, without the parameter overhead of a deeper single-kernel stack.

The network begins with a 3×3 stride-2 stem convolution followed by Batch Normalisation and a SiLU non-linearity. Three subsequent stages repeat the multi-kernel block, interleaved with max-pool layers for progressive spatial reduction ($96 \rightarrow 48 \rightarrow 24 \rightarrow 12 \rightarrow 6$). A global adaptive-average-pool collapses the final 6×6 feature map to 1×1, after which a single linear layer produces the 10-way class logits. To stabilise training on small data regimes we incorporate Batch Normalisation throughout, and insert a Dropout-2d layer ($p = 0.2$) before the classifier to encourage feature-level regularisation. The network is trained with the **binary-cross-entropy** loss, whereas **accuracy** is used as the objective during hyper-parameter search so that results remain directly comparable between the CNN and the classical machine-learning baselines. The following hyper-parameters are tuned with Bayesian optimisation using uniform priors:

- **Learning rate** – A higher value accelerates learning but risks unstable (divergent) updates; a lower value slows progress while promoting smoother convergence.
- **Adam momentum** (β_1) – Adam maintains an exponential moving average of past gradients. Larger β_1 values smooth updates and reduce variance, but may impede the optimiser's ability to escape shallow local minima.
- **Convolutional kernel width** – Changing the kernel size alters the receptive field and hence the scale of features the network can capture.
- **Batch size** – Mini-batch updates approximate stochastic gradient descent. Larger batches yield smoother gradient estimates (similar in effect to momentum) but can obscure informative sample-level variations.
- **Epochs** – The number of complete passes over the training set. Because optimisation is stochastic, more (or fewer) epochs may be required to reach a stable optimum, depending on the other hyper-parameters.

1.8 Hyper-parameter optimisation

Model hyper-parameters are selected on a stratified $20\ \%$ subsample of the training data. Discrete grids are searched exhaustively using 5-fold cross-validation for small spaces, whereas larger or continuous spaces are explored with Gaussian process Bayesian optimization, using SciKit Optimizes BayesSearchCV. The candidate ranges were:

- **Random Forest**
  - Number of trees $n_{\text{estimators}} \in 10, 11, \ldots, 50$
  - Maximum depth $\in 5, 11, \ldots, 50$
  - Minimum samples to split $\in 2, 3, \ldots, 20$
- ***k*NN** (Bayesian optimisation)
  - Number of neighbours $k \in 50, 51, ...100$
  - Weighting scheme $\in$ uniform, distance
  - Minkowski power $p \in 1, 2$
- **Bag of Words**
  - Vocabulary size $k \in 50, 100, 200$
- **Convolutional Neural Network**
  - Kernel width $\in 3, 5, 7$
  - Learning rate $\in [10^{-3}, 10^{-1}]$
  - Momentum $\in [0.80, 0.99]$
  - Epochs $\in 2, 3, ...20$
  - Batch size $\in 4, 8, 16$

1.9 Accuracy statistics

Performance is reported both at dataset level and per class using two complementary metrics. Overall accuracy is the proportion of correctly classified samples, while class-wise confusion statistics are aggregated in a confusion matrix $\mathbf{C}$ whose entry $Cj\ell$ counts how often a sample of true class j is predicted as ℓ. Class-balanced measures such as precision, recall, and F_1 can thus be derived if required.

2 Results

2.1 Hyperparameter Optimization Classical CV

Table 2.1 lists the top three combinations of classifier, normalization, and feature set—ranked by mean training-set accuracy—on the STL-10 and Mammals datasets. Across both domains, a Random Forest (RF) trained on dense keypoint descriptors led the pack ($\mu = 0.409$). While k-NN yielded respectable results, RF's ability to learn its own distance metric suggests that it can better exploit complex feature relationships.

On STL-10, SIFT descriptors outperformed raw-pixel histograms, implying that orientation-aware features promote tighter clustering and easier separation in feature space. Conversely, for Mammals the dense pixel histogram edged out SIFT, hinting that inter-species differences in greyscale intensity patterns may be more salient than edge-based orientations.

Finally, vocabulary size (the number of visual-word centroids K) had only a minor effect when $K \in 100, 200$, but dropped noticeably at $K = 50$. This indicates that too small a codebook lacks the expressiveness needed for either classification task.

Table 1: Top 3 accuracy split by dataset average values come from 3-fold cross-validation on subset of training data.

Dataset	Train Acc.	Std	Norm	Feature	K	Classifier
STL10	0.409	0.028	Min-Max	dense & SIFT	100	RF
	0.353	0.003	Hist Eq.	dense & SIFT	200	k-NN
	0.339	0.016	Min-Max	dense & DH	200	RF
Mammals	0.220	0.015	Min-Max	dense & DH	100	RF
	0.203	0.012	Hist Eq.	dense & DH	200	k-NN
	0.201	0.018	Min-Max	SIFT & DH	100	RF

2.2 Hyperparameter Optimization CNN

On both datasets the architecture that fuses three kernel sizes (3, 5, and 7) outperforms the single-kernel "Net" by a substantial margin (STL-10: 0.645 vs 0.478; Mammals: 0.571 vs 0.293). Having multiple receptive fields appears to help the model capture both fine and coarse structures. The colour images improved the performance of the classical Net for both STL10 (0.478 vs 0.471) and for Mammals (0.293 vs 0.222), indicating that in natural scenes, chromatic cues carry discriminative information that a CNN can exploit. A small batch appears to improve the performance of the network on the training set, indicating that shallow nets can avoid overfitting in low sample scenarios even with increased gradient updates.

Table 2: Top 3 accuracy split by dataset average values come from 3-fold cross-validation on subset of training data.

Dataset	Train Acc.	Std	Colour	Batch	LR	β_1	Kernel	Classifier
STL10	0.645	0.036	RGB	4	$1.08e^{-2}$	0.84	3,5,7	MultiScaleNet
	0.478	0.019	RGB	16	$3.71e^{-2}$	0.44	3	Net
	0.471	0.028	Greyscale	4	$1e^{-4}$	0.99	3	Net
Mammals	0.571	0.009	RGB	4	$1.30e^{-2}$	0.89	3,5,7	MultiScaleNet
	0.293	0.011	RGB	4	$2.58e^{-3}$	0.73	3	Net
	0.222	0.014	Greyscale	16	$2.62e^{-2}$	0.49	3	Net

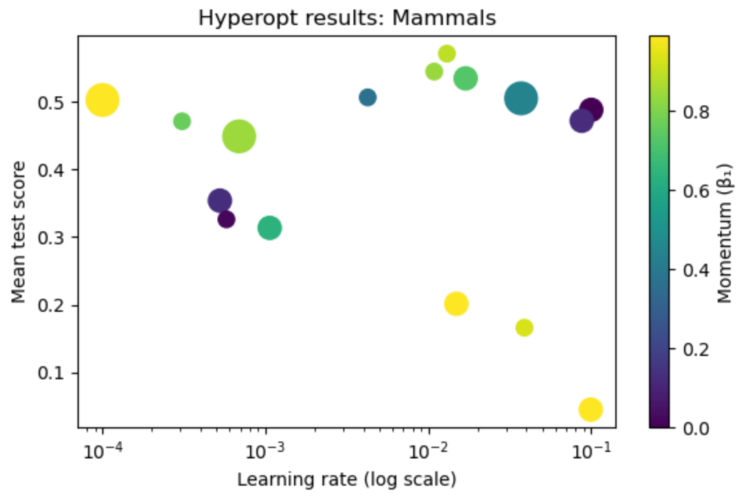

Figure 1: Hyperparameter search results for the Mammals dataset (3-fold CV). Each point represents a single configuration, plotted by learning rate (log scale) vs. mean test accuracy. Bubble size $\propto$ batch size, and color encodes momentum (β_1). Optimal performance clusters at lr $\sim 1 \times 10^{-2}$, $\beta_1 \sim [0.8, 0.9]$, and batch size = 4.

Figure 1 visualizes the effect of learning rate, momentum, and batch size on 3-fold cross-validated accuracy for the Mammals dataset for the top performing model. Each point plots one hyperparameter configuration's mean test score against its learning rate (log scale), with bubble size proportional to batch size and color indicating momentum (β_1). We observe a clear "sweet spot" of learning rates around $5 \times 10^{-3}, 2 \times 10^{-2}$: configurations in this range achieve the highest accuracies ($\approx 0.53 - -0.57$). Likewise, momentum values in the mid-range ($\approx 0.8, 0.9$) consistently yield top scores, while extremes at 0 or near 1 correlate with poor convergence. Taken together, these results recommend tuning around $LR \approx 1 \times 10^{-2}$, $\beta_1 \approx 0.85$, and batch size = 4 to maximize performance on the Mammals classification task. Although the relationship between hyper parameters is highly non-linear and strongly suggests they must be jointly optimized for maximal performance.

2.3 Performance Full Dataset

Table 2.3 summarizes, for both STL-10 and Mammals, the accuracy of each normalization scheme, feature descriptor, and keypoint strategy on the full datasets using their respective best classifiers. Notably, when trained on the entire data, raw-pixel inputs outperformed their cross-validated results from the hyperparameter search—highlighting that some classifiers can leverage raw intensities more effectively as sample size grows. That said, pixel-based training is also the costliest, and the top-performing configuration remained dense SIFT descriptors, which consistently surpassed raw pixels in accuracy. Sparse SIFT keypoints appeared only once among the top three models (Mammals: $\mu = 0.234$), suggesting that its standard contrast and edge thresholds may miss valuable features; tuning these parameters could yield further gains.

For the CNN approximately the same performance observed for the hyperparameter optimization is observed for the full test data performance 2.3. However notably the performance of the greyscale CNN improved compared to its performance on the training data by a substantive margin on the mammals dataset (0.271 vs 0.222) potentially suggesting that data quantity was a bottle neck to the nets performance on the task.

To compare the classical CV pipeline (Random Forest on dense SIFT features) against our MultiScaleNet CNN on STL-10, we compute confusion matrices for each best-performing model (Figure 2).

Table 3: Top 3 accuracy for testing data split showing normalisation scheme (Norm), the keypoint & detector algorithm (Feature), number of cluster centers (K) and the classifier type.

Dataset	Test Acc.	Norm	Feature	K	Classifier
STL10	0.415	Min-Max	dense & SIFT	100	RF
	0.400	Min-Max	None & None	0	RF
	0.379	Hist Eq.	dense & SIFT	200	KNN
Mammals	0.250	Min-Max	dense & DH	100	RF
	0.238	Min-Max.	None & None	0	RF
	0.234	Min-Max	SIFT & DH	100	RF

Table 4: Top 3 accuracy split by dataset average values come from 3-fold cross-validation on subset of training data.

Dataset	Test acc.	Colour	Batch	LR	β_1	Kernel	Classifier
STL10	0.665	RGB	4	$1.08e^{-2}$	0.84	3,5,7	MultiScaleNet
	0.509	RGB	16	$3.71e^{-2}$	0.44	3	Net
	0.492	Greyscale	4	$1e^{-4}$	0.99	3	Net
Mammals	0.581	RGB	4	$1.30e^{-2}$	0.89	3,5,7	MultiScaleNet
	0.314	RGB	4	$2.58e^{-3}$	0.73	3	Net
	0.271	Greyscale	16	$2.62e^{-2}$	0.49	3	Net

Classical CV (RF + SIFT) achieves reasonable accuracy on rigid-body classes—Airplane (0.59), Car (0.55), Ship (0.42), Truck (0.50)—but struggles with soft-bodied animals. Bird, Cat, Deer, Dog, Horse, and Monkey score 0.39, 0.19, 0.41, 0.16, 0.51, and 0.42, respectively. The most frequent confusions occur among four-legged mammals: Cat$\rightarrow$Deer (166), Cat$\rightarrow$Monkey (161), Dog$\rightarrow$Horse (166), and Dog$\rightarrow$Monkey (181). This suggests that, with only greyscale edge-based descriptors, the model over-relies on shared outlines and texture-less gradients, failing to exploit subtle shape or color cues.

MultiScaleNet (CNN) markedly improves upon these soft-body errors while boosting—rigid-body performance: Airplane (0.79), Car (0.82), Ship (0.80), Truck (0.70). Soft-bodied accuracies rise to Bird 0.55, Cat 0.51, Deer 0.62, Dog 0.59, Horse 0.65, Monkey 0.60. Importantly, many of the same "outline" confusions persist (e.g. Bird $\rightarrow$ Monkey, Dog $\rightarrow$ Horse), implying that—even with multi-scale convolutional filters—the network sometimes defaults to coarse silhouette features. However, the CNN also introduces novel error modes: for instance, Cat $\rightarrow$ Dog misclassifications increase relative to RF, perhaps reflecting the model's partial learning of facial or fur-pattern cues that classical SIFT histograms cannot capture.

Interpretation. Both pipelines exhibit a failure mode on fine-grained four-legged classes. The CNN's superior performance on soft bodies demonstrates that learnable, multi-scale filters can extract richer, color-dependent cues, but further improvements may require explicit attention to part-based features (e.g. heads vs. limbs) or integration of color-texture augmentations. One further simulation would be to attempt to train a model that can discriminate and cluster images without labels, thus maximally utilizing latent image features, prior to attempting the classification task. Addressing these remaining confusions could close the gap toward human-level granularity on animal classes.

3 State of the art

3.1 Classical Algorithms in Computer Vision

Prior to the deep-learning era, robots relied on classical computer-vision pipelines built from hand-crafted feature detectors, descriptors and geometric estimation methods—entirely foregoing large annotated datasets. Given a monocular or stereo camera, a robot would detect point features (e.g. corners, blobs or edges) in each frame and match these keypoints across views to recover three-dimensional structure [4]. Landmark contributions such as the Harris corner detector (1988)[5] and Lowe's Scale-Invariant Feature Transform (SIFT, 2004)[2] introduced repeatable, invariant keypoints whose descriptors remain distinctive under changes in scale, rotation and illumination. By chaining these matches with robust estimators like RANSAC [6] and exploiting epipolar geometry, researchers assembled end-to-end systems for tasks such as Simultaneous Localisation and Mapping (SLAM) or in object recognition and manipulation. Nevertheless, these approaches exhibit two fundamental limitations. First, keypoints based systems tend to falter in low-texture environments or under severe appearance changes, where stable keypoints are scarce. Second, representing a scene purely as clusters of corners or edges does not incorporate semantic understanding, hindering tasks such as object-level reasoning, dynamic scene interpretation, monocular depth estimation or pixel-wise segmentation. Attempts

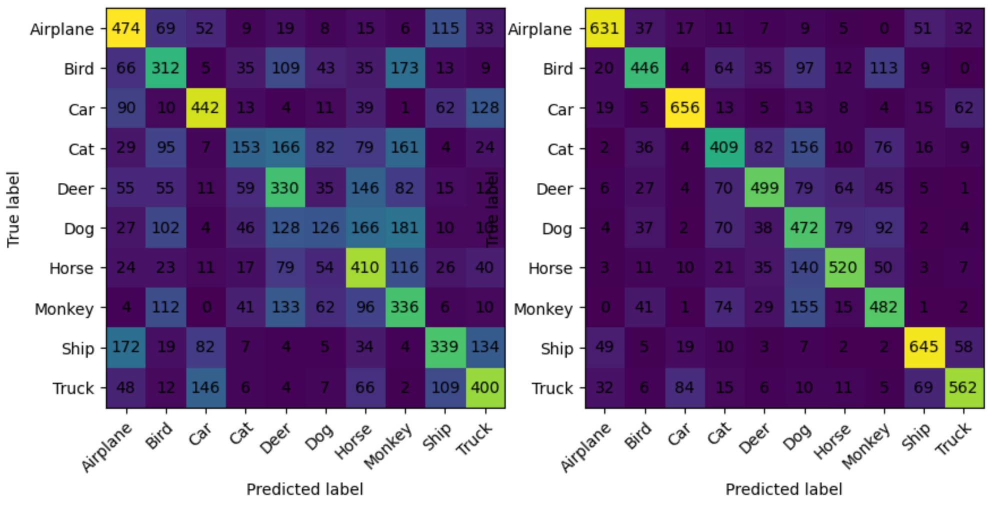

Figure 2: Left: Confusion matrix computed for STL10 dataset with the RF classifier, $K = 100$, Features: dense and SIFT, Norm: Min-Max, Test Acc: 0.415. Right: Confusion matrix computed for STL10 dataset with CNN classifier (MultiScaleNet), LR = $1.08e^{-2}$, batch = 4, β_1 = 0.84, colour: RGB

to inject semantics via bag-of-visual-words models[3] have achieved only limited success, partially because they discard spatial relationships among descriptors.

3.2 Deep Learning Algorithms in Computer Vision

The 2012 breakthrough of deep convolutional neural networks (CNNs) for ImageNet classification inaugurated a shift from manual feature engineering to data-driven representation learning[7]. Rather than designing corner detectors or gradient-histogram descriptors by hand, one trains a CNN on millions of labeled images (e.g. ImageNet, COCO [8]) so that it automatically learns hierarchical feature extractors—from edges and textures in early layers to object parts and categories in deeper layers [9].

Deep learning has also been applied to SLAM and visual odometry. Learned front-ends—such as SuperPoint [10] keypoint detectors and descriptors—offer improved repeatability in low-texture scenes, while CNN-based global descriptors (e.g. NetVLAD) outperform traditional bag-of-words methods for place recognition and loop-closure detection under drastic appearance changes[11]. Researchers have further explored end-to-end learning of camera motion and depth from monocular sequences, using self-supervision to teach a network to predict both pose and per-pixel depth without explicit geometric solvers [12]. Although fully learned SLAM remains an open challenge due to reliability concerns, hybrid approaches—where deep networks serve as auxiliary modules (e.g. depth [13] or semantic predictors[14]), module replacements (e.g. learned feature extractors) or, in the most ambitious cases, as unified pose-and-map estimators—are already improving performance on many fronts[15, 16].

In robotic manipulation, deep learning and reinforcement learning have supplanted traditional pipelines that required 3D object models and hand-tuned planners[17]. By training policies directly on raw camera images and reward signals, robots learn end-to-end grasping strategies without explicit camera calibration or object modeling[17, 18]. This self-supervised approach enables continuous skill acquisition and task generalization, provided a suitable reward function can be defined. One major advantage of self-supervised learning is that it can provide models with the capacity to make temporal predictions, i.e. in autonomous driving cars that are trained to predict next frame given a steering input. By leveraging future-frame prediction as an intrinsic training signal, these models not only close the perception–action loop in real time but also lay the groundwork for more anticipatory, robust control policies in ever more complex and dynamic environments[19]. Although deep learning has a wide range of applications in computer vision there are several limitations that represent active areas of research in order to increase the adoption of NN for robotics and computer vision.

3.3 Limitations

Data Requirements. Classical pipelines operate deterministically on any visual input and require no prior training—an advantage when deploying robots in novel environments. In contrast, deep networks typically demand large labeled datasets to reach peak performance. Mitigation strategies include:

- *Self-Supervised Learning*, which derives supervisory signals from the data itself (e.g. photometric consistency across frames).
- *Transfer Learning*, whereby a model pre-trained on a large, generic corpus is fine-tuned with relatively few task-specific examples.
- *Simulation (Virtual Worlds)*, where physics-based environments generate synthetic images and annotations at scale; robots can query the simulator to learn object detection or navigation policies before deployment in the real world.

Computation and Latency. Classical feature detectors and geometric solvers are engineered for real-time CPU execution, whereas deep models can incur prohibitive inference times on resource-constrained platforms. Common remedies include:

- *Hardware Acceleration*, such as on-board GPUs, FPGAs or dedicated NPUs.
- *Efficient Architecture Design*, including network pruning, quantization and knowledge distillation to shrink model size with minimal accuracy loss.
- *Hybrid Pipelines*, in which fast, classical pre-filters (e.g. cascade detectors) identify regions of interest that are then processed by slower, learned modules.

Robustness and Interpretability. Classical methods fail predictably when their geometric assumptions break (e.g. in textureless or motion-blurred scenes). Deep networks, by contrast, often produce overconfident but incorrect outputs under distributional shifts[20], and their internal representations remain largely opaque. Ongoing research in *explainable AI* aims to establish uncertainty estimates and interpretability tools, which are crucial for safety-critical robotic applications [21].

Sim-to-Real Gap. Models trained solely in simulation may underperform when faced with the visual and physical complexities of the real world—lighting extremes, sensor noise or unmodelled dynamics [22]. Techniques to bridge this gap include:

- *Domain Randomization*, which augments synthetic data with wide variations in textures, lighting and object appearances so that real-world conditions appear as just another variation [23].
- *Mixed Reality Fine-Tuning*, wherein a mostly simulated dataset is supplemented with a smaller, curated set of real images to adapt the model to real-world stimuli.

3.4 Deep Learning Architectures

Below is a concise overview of prominent deep architectures in modern computer vision—this list is illustrative rather than exhaustive:

1. **Convolutional Neural Networks (CNNs).** CNNs employ learned convolutional kernels and pooling layers to extract translation-invariant features while preserving spatial structure. Architectures such as ResNet [24] introduce residual (skip) connections that enable effective training of very deep networks by learning only the residual signal at each layer.
2. **Vision Transformers (ViTs).** Adapting the transformer model from NLP, ViTs partition an image into fixed-size patches and apply self-attention to capture long-range dependencies across the entire scene [25]. By modeling global context from the outset, ViTs can outperform CNNs on large-scale tasks—though their higher computational cost and need for substantial training data remain challenges for real-time robotics.
3. **Self-Supervised and Generative Models.** Autoencoders [26], contrastive frameworks (e.g. SimCLR [27], MoCo [28]) and diffusion [29] or flow-matching models [30] learn rich representations without explicit labels by reconstructing or contrasting input data. These methods not only facilitate downstream fine-tuning with limited annotations but also enable controlled image synthesis via conditioning (e.g. generating segmentations or novel views).

References

[1] Coates, A., Ng, A. & Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 215–223 (JMLR Workshop and Conference Proceedings, 2011).

[2] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* **60**, 91–110 (2004).

[3] Sivic & Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, 1470–1477 vol.2 (2003).

[4] Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision*, 1403–1410 (IEEE, 2003).

[5] Harris, C., Stephens, M. *et al.* A combined corner and edge detector. In *Alvey Vision Conference*, vol. 15, 10–5244 (Citeseer, 1988).

[6] Fischler, M. A. & Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM* **24**, 381–395 (1981).

[7] Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* **25** (2012).

[8] Lin, T.-Y. *et al.* Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part v 13*, 740–755 (Springer, 2014).

[9] Zeiler, M. D. & Fergus, R. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, 818–833 (Springer, 2014).

[10] DeTone, D., Malisiewicz, T. & Rabinovich, A. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 224–236 (2018).

[11] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T. & Sivic, J. NetVLAD: CNN architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5297–5307 (2016).

[12] Zhou, T., Brown, M., Snavely, N. & Lowe, D. G. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1851–1858 (2017).

[13] Tateno, K., Tombari, F., Laina, I. & Navab, N. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6243–6252 (2017).

[14] McCormac, J., Handa, A., Davison, A. & Leutenegger, S. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 4628–4635 (IEEE, 2017).

[15] Czarnowski, J., Laidlow, T., Clark, R. & Davison, A. J. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters* **5**, 721–728 (2020).

[16] Macario Barros, A., Michel, M., Moline, Y., Corre, G. & Carrel, F. A comprehensive survey of visual slam algorithms. *Robotics* **11**, 24 (2022).

[17] Kalashnikov, D. *et al.* Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, 651–673 (PMLR, 2018).

[18] Levine, S., Finn, C., Darrell, T. & Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* **17**, 1–40 (2016).

[19] Finn, C., Goodfellow, I. & Levine, S. Unsupervised learning for physical interaction through video prediction. *Advances in neural information processing systems* **29** (2016).

[20] Guo, C., Pleiss, G., Sun, Y. & Weinberger, K. Q. On calibration of modern neural networks. In *International Conference on Machine Learning*, 1321–1330 (PMLR, 2017).

[21] Das, A. & Rad, P. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371* (2020). `2006.11371`.

[22] Zhao, W., Queralta, J. P. & Westerlund, T. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 737–744 (IEEE, 2020).

[23] Tobin, J. *et al.* Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23–30 (IEEE, 2017).

[24] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778 (2016).

[25] Dosovitskiy, A. *et al.* An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020). `2010.11929`.

[26] Hinton, G. E. & Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science* **313**, 504–507 (2006).

[27] Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 1597–1607 (PmLR, 2020).

[28] He, K., Fan, H., Wu, Y., Xie, S. & Girshick, R. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9729–9738 (2020).

[29] Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems* **33**, 6840–6851 (2020).

[30] Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M. & Le, M. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747* (2022). `2210.02747`.

A Code for Parts 3 and 4

All code is available at `https://github.com/Detriatis/computer_vision`.

Below is the main pipeline driver (‘main/pipeline/cv_fit.py‘). The rest of the modules it imports can be found in the GitHub repo.

Listing 1: Classic Pipeline

```python
import numpy as np
import pandas as pd
from collections import OrderedDict
from pathlib import Path

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

from main.utils.scikit_interface import ImageNormalizer, DescriptorExtractor,
    BoWTransformer, get_sklearn_metric
from main.utils.image_loading import load_directory_images
from main.utils.cv_dataclasses import Images


def class_label_assigments(labels, kps, accuracy_scores):
    label_dic = {str(label): {'n_keypoints': 0, 'accuracy': 0} for label in sorted
        (np.unique(labels))}
    if kps is not None:
        for kp, label in zip(kps, labels):
            label_dic[str(label)]['n_keypoints'] += len(kp)

    for i, score in enumerate(accuracy_scores):
        label_dic[str(i + 1)]['accuracy'] = score
    return label_dic




def pipeline(clf, imgs: Images, row = None, bowtransformer = None, train: bool =
    True):
    X = imgs.imgs
    if row is not None:
        try:
            norm = int(row['norm_type'])
        except ValueError:
            norm = str(row['norm_type'])
        desc_algo = row['descriptor_algo']
        kp_algo = row['keypoints_algo']
        if kp_algo != 'none':
            k = int(row['k'])

    imagenormalizer = ImageNormalizer(norm).fit(X)
    X = imagenormalizer.transform(X)

    descriptorextractor = DescriptorExtractor(kp_algo=kp_algo, desc_algo=desc_algo
        ).fit(X)
    X = descriptorextractor.transform(X)
    if kp_algo != 'none':
        if train:
            bowtransformer = BoWTransformer(k=k).fit(np.concat(X))

        bows = bowtransformer.transform(X)
    else:
        bows = np.array(X).reshape(-1, 96 * 96)

    if train:
        clf.fit(bows, imgs.labels)

    preds = clf.predict(bows)

    return clf, preds, bowtransformer, descriptorextractor

DATA_PARAMETERS  = {
    'directorypath': Path("/home/REDACTED/Projects/manchester/computer_vision/datasets/processed/"),
    'bestclassifierpath': "/home/REDACTED/Projects/manchester/computer_vision/results/hyperopt/aggregated_results/cv_best_classifiers.csv",
    'writeoutpath': "/home/REDACTED/Projects/manchester/computer_vision/results/"
                    }

classifiers = {
        'rf' : RandomForestClassifier(),
        'knn' : KNeighborsClassifier()
}

if __name__ == '__main__':
    test_bool = False

    classifiers_df = pd.read_csv(DATA_PARAMETERS['bestclassifierpath'])

    feature_params = {}
    norm_params = {}
    bow_params = {}
    classifier_params = {}
    rows = []

    for i in range(len(classifiers_df)):
        row = classifiers_df.iloc[i].copy()

        imgs = load_directory_images(DATA_PARAMETERS['directorypath'] / row.dataset, train=True)
        test_imgs = load_directory_images(DATA_PARAMETERS['directorypath'] / row.dataset, train=False)

        accuracy = get_sklearn_metric('accuracy')

        classifier = row['classifier']
        classifier_params =  eval(row['params'], {'OrderedDict' : OrderedDict})

        clf = classifiers[classifier]
        clf = clf.set_params(**classifier_params)

        if not test_bool:
            clf, train_preds, bowtransformer, _ = pipeline(clf, imgs, row, train=True)
            clf, test_preds, _, descriptorextractor =  pipeline(clf, test_imgs, row, bowtransformer=bowtransformer, train=False)

            kps_list = descriptorextractor.kps
            descs_list = descriptorextractor.descs

            train_score = accuracy(imgs.labels, train_preds)
            test_score = accuracy(test_imgs.labels, test_preds)

            cm = confusion_matrix(test_imgs.labels, test_preds)
            results = class_label_assigments(test_imgs.labels, kps_list, (cm.diagonal() / cm.sum(axis=1)))

            results_df = pd.DataFrame(results)

            print(f'Score on training set {train_score} with classifier {classifier}')
            print(f'Score on testing set {test_score} with classifier {classifier}')
            row['fullmodelaccuracy'] = test_score
            row['index'] = i
            rows.append(row)


            outpath = Path('/home/REDACTED/Projects/manchester/computer_vision/
                results/fullmodel', f'{classifier}_{str(i)}')
            outpath.mkdir(parents=True, exist_ok=True)
            results_df.to_csv(outpath / 'class_accuracy.csv', index=False)
            np.savetxt(outpath / "confusion_matrix.csv", cm, delimiter=",")

    rowsframe = pd.concat(rows, axis=1).T
    rowsframe.to_csv(outpath.parent / 'cv_fullmodel.csv', index=False)
```

Listing 2: Classic HyperoptPipeline

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from main.utils.scikit_interface import ImageNormalizer, DescriptorExtractor,
    BoWTransformer, SKlearnPyTorchClassifier
from main.pipeline.cnn import Net
from main.utils.image_loading import load_directory_images

import logging
from joblib import Memory

from skopt.space import Real, Integer, Categorical
from skopt import BayesSearchCV

import cv2
import pandas as pd
import numpy as np
from pathlib import Path
import multiprocessing as mp

mp.set_start_method('forkserver', force=True)

# Norm Paramaters
search_space_norms = [
            cv2.NORM_MINMAX,
            'eq_hist',
            ]

kp_desc = [
    ('none', 'none'),
    ('sift', 'sift'),
    ('sift', 'dense_hist'),
    ('dense', 'sift'),
    ('dense', 'dense_hist'),
]

detector_params = {
    'sift' : {},
    'dense': {
        'kp_size': 1,
        'step_size': 5,
    },
    'none': {},
}
# Bag of Words Parameters
search_space_bow = {
    'k': 0,
    'max_iters':1000,
    'batch':1024,
}
words = [50, 100, 200]


# Classifier Parameters
search_space_rf = {
        'n_estimators': Integer(50 ,500),
        'max_depth': Integer(5, 50),
        'min_samples_split': Integer(2, 20),
}

search_space_knn = {
        'n_neighbors': Integer(10, 200),
        'weights': Categorical(['uniform', 'distance']),
        'p': Integer(1, 2),
}


search_space = {

    'rf': {
        'classifier' : RandomForestClassifier(),
        'parameter_space' : search_space_rf
            },

    'knn': {
        'classifier' : KNeighborsClassifier(),
        'parameter_space' : search_space_knn
            },

  }

directories = ['mammals', 'stl10']
for dir in directories:
    print(dir)
    DIRECTORYPATH = Path("/home/REDACTED/Projects/manchester/computer_vision/
        datasets/processed/") / dir
    WRITEOUTPATH = "/home/REDACTED/Projects/manchester/computer_vision/results"

    imgs = load_directory_images(DIRECTORYPATH, train = True, subsample=100)
    X = imgs.imgs
    print(f'Training on {len(X)}')

    for norm_type in search_space_norms:

        imagenormalizer = ImageNormalizer(norm_type=norm_type).fit(X)
        norm_x = imagenormalizer.transform(X)

        for kp, desc in kp_desc:
            descriptorextractor = DescriptorExtractor(desc_algo=desc, kp_algo=kp,
                **detector_params[kp]).fit(norm_x)
            transform_x = descriptorextractor.transform(norm_x)

            for k in words:
                if kp != 'none':
                    search_space_bow['k'] = k
                    bowtransformer = BoWTransformer(**search_space_bow).fit(np.
                        concat(transform_x))
                    bows = bowtransformer.transform(transform_x)
                else:
                    k = 'None'
                    bows = np.array(transform_x).reshape(-1, 96 * 96)

                for key, value in search_space.items():

                    print(norm_type, desc, kp, k, key, dir)
                    classifier = value['classifier']
```

```python
                    parameter_space = value['parameter_space']

                    opt = BayesSearchCV(
                        classifier,
                        parameter_space,
                        n_iter=16,
                        scoring='accuracy',
                        cv=3,
                        verbose=1,
                        random_state=1,
                        n_jobs=-1,
                        n_points=1
                    )
                    opt.fit(bows, imgs.labels)

                    result_df = pd.DataFrame(opt.cv_results_)[[
                        'params', 'mean_test_score', 'std_test_score'
                        ]].sort_values('mean_test_score', ascending=False)
                    result_df['norm_type'] = norm_type
                    result_df['descriptor_algo'] = desc
                    result_df['keypoints_algo'] = kp
                    result_df['k'] = k
                    result_df['classifier'] = key
                    result_df['dataset'] = dir

                    iterative_path = Path(WRITEOUTPATH, 'hyperopt', 'final_iterative_exps.csv')
                    if iterative_path.exists():
                        df = pd.read_csv(iterative_path)
                        iterative_df = pd.concat([result_df, df])
                        iterative_df.to_csv(iterative_path, index=False)
                    else:
                        result_df.to_csv(iterative_path, index=False)
                print(kp)
                if kp == 'none':
                    print('breaking loop')
                    break
```