# CM32022/52035 – Advanced Computer Vision Coursework 1 2025/26

Dr Da Chen and Dr Davide Moltisanti

| | |
|---|---|
| **Set** | 29/10/2025 (week 5) |
| **Due** | 16/03/2026 (week 25), 8 pm |
| **Percentage of overall unit mark** | 50% |
| **Submission Location** | Moodle |
| **Submission Components** | Code and report |
| **Submission Format** | `.pdf` file named 'Group_XX_report.pdf'<br>`.pdf` file named 'Group_XX_GCF.pdf'<br>`.zip` file named 'Group_XX_code.zip' |
| **Anonymous Marking** | No |
| **Generative AI Assessment Categorisation** | Type A (guidance) |

## 1 Overview

There are 7 mandatory tasks and 3 optional ones in this coursework. The first 3 tasks are based on classic computer vision methods and are all required in this coursework. The remaining 7 tasks (4 required, 3 optional) are based on deep learning approaches. The aim is to obtain an understanding of solving computer vision problems.

### 1.1 Classic computer vision

- In the first task, the aim is to use concepts from edge detection and line fitting to be able to measure angles for a pair of lines.
- In the second task, the aim is to use filtering, convolution and template matching to obtain intensity based template matching.
- In the third task, the aim is to implement feature matching with SIFT/SURF features to match the templates using an interest point-based object matching routine.

### 1.2 Deep-learning-based computer vision

With these tasks you will take a modern learning-based approach to address a standard computer vision problem: classification. You will also implement a ranking method based on the triplet loss, and optionally will work on feature visualisation and zero-shot generalisation.

# 2 Submission details

This coursework is worth 50 points from the overall mark of 100 for this unit. Marks are specified for each task later in Table 3. A report will be your main method of assessment. Your code will be auto-marked based on the output and performance of your implemented solutions. Students should form groups of up to **five** individuals and submit one report per group. Students can choose their partners until **3/11/2025 8 pm**. For more details about group marks, please check the other file named "Group Contribution Explained".

Each student has to ensure the following: your group submits the project report, code (one submission per group), and the Group Contribution Form (GCF). You will need to mark each member's work, and that will determine the final mark of each team member. If there are fewer than five members, you still need to ensure that you provide peer-marking for your group and submit the GCF. *Note that there is no advantage in this coursework for fewer group members*. Thus, it is **highly recommended** that students form groups of four or five students. It is your responsibility to check that you correctly submit your work. Once you have submitted to Moodle, you should download your submission and **verify it**.

The submission deadline for your report, code, and GCF is **16/03/2026 8 pm**. Your submission should be uploaded to Moodle and should consist of the report PDF, the GCF PDF, and the code and models as a zip file.

The main part of your report should not exceed the 9,000 word limit. After the main part, please attach the references and appendices, if any (these are excluded from the word count). In addition, you should upload all the code relevant to your implementations in order to evaluate them against an independent test set. Code must be well commented (indicating major steps). You should ensure that it is possible for us to run your code on additional test samples. This implies that the code is self-contained without any absolute image paths, and any appropriate Readme file with environment setup, if required, should be provided.

Your report should demonstrate and evaluate your results. It must include figures of appropriate resolutions. Evaluation can be qualitative (how it looks) and, wherever possible, quantitative (tested numerically). You should also concisely explain your approach to solve each task, the choices you make (e.g. hyperparameters, etc), and answer questions in each part. The marks will be decided based on the marking guideline specification that we will follow for this unit.

### Late submissions

The university policy will be followed for late submissions. If a piece of work is submitted after the submission date, the maximum possible mark will be 40% of the full mark. If the work is submitted more than five working days after the submission date, the student will receive zero marks. **Requests for extensions should be made to the Director of Studies**. Lecturers and tutors cannot approve extensions. Please make sure you are familiar with the department's coursework deadline extension policy.

Figure 1: Measure the angle between two lines.

# 3 Tasks

## Classic computer vision methods (C)

### 3.1 Task C1: Measuring the angle

In this task, given an image with a pair of lines, we need to ensure that we can measure the smaller angle formed by the two lines. An example image is provided in Figure 1. This should be obtained irrespective of the position of the lines in the image.

The main ideas you can use to solve the problem are edge detection and line fitting. You would be expected to explore line fitting on the edges, and subsequently, you should measure the angle through the appropriate code. You should demonstrate that you are able to solve the task in a variety of situations. A set of example images is provided for this task.

Analysis: You should provide an analysis with respect to the choice of edge detection function and the range of parameters used for line fitting.

### 3.2 Task C2: Intensity-based template matching

In this task, you are provided with a set of training images that consist of icons of objects. The set of training icons is provided as shown in Figure 2. In this task, you will implement algorithms for matching images, similar to the principle of playing Dobble.

Output: Detect objects in the Test images, recognise to which class (from the 50 Training image classes) they belong, and identify their bounding boxes. An example of a test image is provided in Figure 3a. To visually demonstrate this, the function should open a box around each detected object and indicate its class label. In your report, include a few examples illustrating the detection of your algorithm for a few images.

Evaluation: Evaluate your algorithm on all Test images to report the overall Intersection over Union (IoU), False Positive (FPR), True Positive (TPR) and Accuracy (ACC) rates, as well as the average runtime. The IOU score of 85% should be used as a threshold to determine

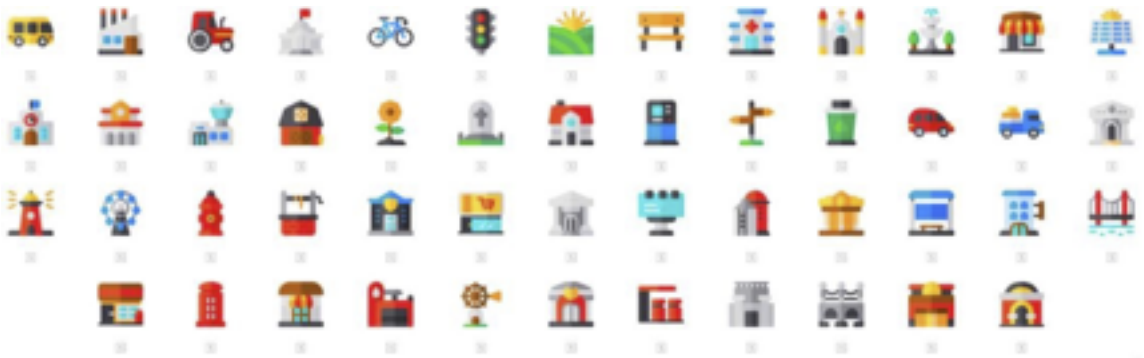

Figure 2: Training images: icons of the village dataset.

(a) (b)

Figure 3: Example of test images for tasks C2 (a) and C3 (b).

true positives and false positives. If the matched template class is truly in the test image and the IOU is above the IOU threshold, then the match is considered a True Positive. Show and explain the cases where this scheme finds difficulty in performing correctly. Give insight about the complexity (memory and runtime) of your algorithm in terms of the problem size, e.g. how does the complexity scale in terms of the number of image pixels, pyramid scales?

Creating scaled templates: You should create a Gaussian pyramid of appropriate scale levels for each RGB training image. This task includes implementing appropriate Gaussian blurs and sub-sampling of the training images through a hierarchy. You may use your own convolution algorithm to apply the Gaussian blur, or you may use the library function. A comparison of your own implementation and the library function is encouraged. In your report, explain the steps you take to build your Gaussian/Laplacian pyramids and visually demonstrate an example pyramid for a training image.

Justify your choice of parameters (e.g. Gaussian kernel specs, initial/final scales, number of scale levels, e.g. how many octaves, number of rotations, etc). To justify quantitatively the chosen number of scales, plot or tabulate the overall object detection performance against these parameters and pick parameters that achieve a reasonably good runtime vs. accuracy trade-off.

Pre-processing: For each (scaled) template set the background to zero, and do the same for the Test data. Intensity-based matching (related to output 2): Slide each (scaled and rotated) template over the Test image to measure their similarities across x-y axes. For the intensity-based similarity score-based matching function, explain why this function is suitable for measuring image similarities. Define appropriate thresholds on this similarity score, a proper non-maxima suppression strategy to avoid false positives within the neighbourhood

of the detected objects and return the output in terms of class label and bounding box: [left, top, right, bottom] for each object.

### 3.3 Task C3: Feature matching using SIFT or equivalent features

Write a function that takes an image from the same dataset for training and testing as in the previous task. One example of a test image is provided in Figure 3b.

Output: Detect objects in the Test images using SIFT or equivalent features (such as SURF), recognise to which class they belong, and identify their scales and orientations. Similar to Task 2, for visual demonstration, the function should open a box around each detected object and indicate its class label. This box is scaled and rotated according to the object's scale and orientation. Demonstrate example images(s) of the outcome detection in your report. Besides, demonstrate example images(s) that show the feature-based matches established between the recognised objects and a Test image, before and after the outlier refinement step.

Evaluation: Evaluate your algorithm on all Test images to report the overall Intersection over Union (IOU), False Positive Rate (FPR), True Positive Rate (TPR) and Accuracy (ACC) rates, as well as the average runtime. The IOU score of 85% should be used as a threshold to determine true positives and false positives. If the matched template class is truly in the test image and the IOU is over the IOU threshold, then the match is considered a True Positive. Otherwise, the match is considered a False Positive. Show and explain cases where this scheme finds difficulty performing correctly. Compare the SIFT/SURF results to those of the task 2 algorithm, e.g. does it improve the overall speed or accuracy? How much? Why? Also, analyse the specific cases where this algorithm does better or worse.

Main steps: You first extract keypoints and feature descriptors from your Test and Training images using the standard SIFT or SURF feature extraction function from a library. Then you match features between images, which will give you the raw noisy matches (correspondences). Now you should decide which geometric transform to use to reject the outliers. Finally, you will define a score on the obtained inlier matches and will use this to detect the objects (icons) scoring high for a given Test image. A basic score is counting the inlier matches, but if you have a better idea report and justify that too.

Similarly, you will have some hyperparameters to tune. This includes the number of Octaves and the (within-octave) Scale levels within SIFT to build scale-spaces for keypoint detection, and the `MaxRatio` parameter within the `matchFeatures` function to reject weak matches. How are these parameters set for this task? Show quantitatively why.

## Deep-learning-based classification (D)

**Dataset** In this part of the coursework you will use a pre-trained CNN to perform various tasks. The tasks are based on the CIFAR 100 dataset [1], which contains 100 classes, with 600 images for each class. Classes are clustered in superclasses, i.e. we have a coarse-grained label (e.g. "flowers") attached to several fine-grained labels (e.g. "orchids, poppies, roses"). Table 1 illustrates the 100 classes grouped into superclasses, while Figure 4 shows a few samples from the dataset. We provide a sub-sampled version of the dataset, split in training and test sets. The training set contains 100 samples per fine-grained class (10,000 images in total) and the test contains 25 samples per fine-grained class (2,500 images in total). The test set used for marking is private and different from the one you will receive (but it is of the same size).

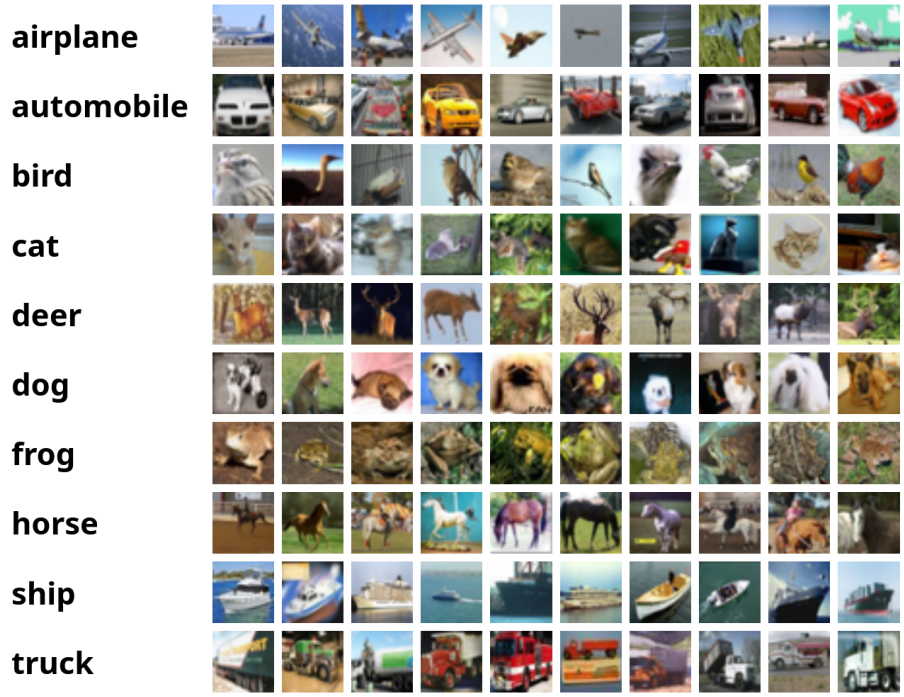


Figure 4: Samples from CIFAR 100. Image from the CIFAR 100 website.

**Model** You will use the MobileNetV3 [2] model pre-trained on ImageNet [3]. You are not allowed to use any other model or pre-training. You will use the model's backbone (i.e. the core of the model before the classification layer) and will have to build your own classifiers. You will have to fine-tune the model (backbone and classifier). You will need to keep the versions of the model that you trained in each task to complete some other tasks. Note that this is a tiny CNN designed for efficiency (it can run on consumer CPUs). We chose this model to make training fast but naturally the size of the model will bring some limitations. Keep this in mind when running your experiments and writing your report.

**Notation** The notation for the D1-D7 tasks is defined in Table 2.

## 3.4 Task D1: Fine-grained classification

Given an image, predict its corresponding fine-grained class label. Formally, given the pre-trained backbone $f$ (i.e. the CNN we provide, which does not have a classification layer), you need to design a classifier $h$ (e.g. a linear layer or a multi-layer perceptron) that takes the backbone's image embedding in input and outputs a class prediction, i.e. $\hat{y}^f = h(f(x))$. You will need to train both $h$ and $f$ using the images and labels in the training set $D^{train}$. For testing, you will need to use the images in the test set $D^{test}$, without feeding the ground truth label $y^f$ to the model. The model will be evaluated by calculating the top-1 accuracy between the predicted class $\hat{y}^f$ and the ground truth label $y^f$ for all images in the test set. In your report, discuss the design choices for your classifier and the performance of your model on the test set.

## 3.5 Task D2: Coarse-grained classification

This task is essentially the same as task D1, with the difference that you will need to predict the coarse-grained class label of an image. Formally, given the pre-trained backbone $f$, you need to design another classifier $h$ that takes the backbone's image embedding in input and

| Superclass | Classes |
|---|---|
| aquatic mammals | beaver, dolphin, otter, seal, whale |
| fish | aquarium fish, flatfish, ray, shark, trout |
| flowers | orchids, poppies, roses, sunflowers, tulips |
| food containers | bottles, bowls, cans, cups, plates |
| fruit and vegetables | apples, mushrooms, oranges, pears, sweet peppers |
| household electrical devices | clock, computer keyboard, lamp, telephone, television |
| household furniture | bed, chair, couch, table, wardrobe |
| insects | bee, beetle, butterfly, caterpillar, cockroach |
| large carnivores | bear, leopard, lion, tiger, wolf |
| large man-made outdoor things | bridge, castle, house, road, skyscraper |
| large natural outdoor scenes | cloud, forest, mountain, plain, sea |
| large omnivores and herbivores | camel, cattle, chimpanzee, elephant, kangaroo |
| medium-sized mammals | fox, porcupine, possum, raccoon, skunk |
| non-insect invertebrates | crab, lobster, snail, spider, worm |
| people | baby, boy, girl, man, woman |
| reptiles | crocodile, dinosaur, lizard, snake, turtle |
| small mammals | hamster, mouse, rabbit, shrew, squirrel |
| trees | maple, oak, palm, pine, willow |
| vehicles 1 | bicycle, bus, motorcycle, pickup truck, train |
| vehicles 2 | lawn-mower, rocket, streetcar, tank, tractor |

Table 1: Classes in CIFAR 100.

| Symbol | What it is |
|---|---|
| $x$ | image |
| $y^f$ | fine-grained class (e.g. “orchids”) |
| $y^c$ | coarse-grained class (e.g. “flower”) |
| $D^{train} = \{(x_i, y_i^f, y_i^c),\ i \in \{1, \ldots, N_{train}\}\}$ | Training set consisting of $N_{train}$ samples |
| $D^{test} = \{(x_i, y_i^f, y_i^c),\ i \in \{1, \ldots, N_{test}\}\}$ | Test set consisting of $N_{test}$ samples |
| $f$ | CNN backbone (before the classification output) |
| $h$ | classifier (e.g. linear layer or multilayer perceptron) |

Table 2: Data and model notation for D1-D7 tasks.

outputs a class prediction, i.e. $\hat{y}^c = h(f(x))$. You will need to train both $h$ and $f$ using the images and labels in the training set $D^{train}$. For testing, you will need to use the images in the test set $D^{test}$, without feeding the ground truth label $y^c$ to the model. The model will be evaluated by calculating the top-1 accuracy between the predicted class $\hat{y}^c$ and the ground truth label $y^c$ for all images in the test set. In your report, discuss the design choices for your classifier and the performance of your model.

## 3.6 Task D3: Coarse-grained and fine-grained classification

Tasks D1 and D2 are single-class classification problems. This task introduces now the multi-label classification problem where your model needs to predict *both* the coarse label *and* the fine label of an image. Given the pre-trained backbone $f$, you will need to design a new classifier $h$ that predicts both $\hat{y}^f = h(f(x))$ and $\hat{y}^c = h(f(x))$. Like in the previous tasks, you will need to train your classifier(s) and the backbone $f$ using the images and labels in the training set $D^{train}$. For testing, you will need to use the images in the test set $D^{test}$, without feeding the ground truth labels $y^f$ and $y^c$ to the model. The model will be evaluated by calculating for all images in the test set: i) the top-1 accuracy between the predicted class $\hat{y}^f$ and the ground truth label $y^f$, ii) the top-1 accuracy between the predicted class $\hat{y}^c$ and the ground truth label $y^c$, iii) the combined top-1 accuracy for both predictions (called "exact match" or "subset accuracy") where for a single image the model's output is correct if *both* predictions are correct. In your report, discuss the design choices for your classifier and the performance of your model on the three metrics.

## 3.7 Task D4: Ranking images

We will now move away from classification and address a more challenging problem: learning to rank images based on the similarity between images. Formally, let:

$$a = (x_a, y_a)$$

be an *anchor* data point from our training set. The job of your model now is to learn a representation that pulls related images close to $a$ and pushes unrelated images further from $a$. The related images are called *positives*, and in our setting we define them as images from the same class as $a$, i.e.

$$p = (x_p, y_p) : y_p = y_a$$

Similarly, unrelated images are called *negatives*, and in our setting we define them as images from a different class, i.e.

$$n = (x_n, y_n) : y_n \neq y_a$$

Your task now is to train the model $f$ (i.e. just the backbone) to embed $x_p$ close to $x_a$ and $x_n$ far from $x_a$, using **the triplet loss** approach. Specifically, the model should learn a representation such that:

$$d\Big(f(x_a) - f(x_p)\Big) + \alpha < d\Big(f(x_a) - f(x_n)\Big)$$

where $d$ is a distance function between tensors (e.g. the Euclidean distance or the Cosine similarity), and $\alpha$ is a scalar hyper-parameter called margin, which you need to tune. To train the model you will need to sample $(a, p, n)$ triplets from the training set $D^{train}$. You will need to run two sets of experiments to form triplets using: i) the fine-grained labels (i.e. $y_i = y_i^f$ above) and ii) the coarse-grained labels (i.e. $y_i = y_i^c$ above). The way you form triplets will influence your model, and in your report you should discuss what sampling method(s) you tried. For testing, your model will be evaluated using the recall@$k$ metric (with $k \in (5, 10, 50, 100)$): for each testing image, you will need to rank the other images in

the test set based on the distance between their embeddings, and calculate the recall based on the $k$ nearest images for each sample, reporting the average. To clarify, given a query image $x_i$ belonging to class $y_i$ you should calculate:

$$R(x_i, k) = \frac{\sum_{j=1}^{k} \mathbb{1}\Big(y_i = \dot{y}(x_j)\Big)}{|y_i|}$$

Where $x_j$ is the $j$-th image nearest to the query $x_i$ (ranked based on their embedding distance), $\dot{y}(x_j)$ is a function that returns its label, and $|y_i|$ is the size of the class. You will need to calculate $R$ as defined above for all images in the test and return the average, for each $k$ value specified above.

Your model will also be evaluated based on the average distance between all valid (anchor, positive) pairs and (anchor, negative) pairs in the test set - the average distance between anchor and positive pairs should be shorter than the distance between anchor and negative pairs. In your report, reflect on the fundamental differences between the triplet loss and the classifier approach from the previous tasks, and discuss the performance of your model (and how it varies with $\alpha$, which you need to tune). If you tried different triplet sampling methods, also discuss how these affect your results. Also, please discuss how results vary between the fine-grained and coarse-grained experiments.

## 3.8 Task D5 (optional): Zero-shot classification

In this task you will test how good your model from D1 is at zero-shot classification. For this task you will need to retrain your model from D1 on a new training-testing split where we dropped 20 fine-grained classes in the training test. This means that there are 20 classes in the test set that your model will not have seen at all during training. You can either reuse the same architecture from D1, or explore some changes that you think might improve the performance of the model in this task. For testing, you will need to implement a $k$-nn classifier using a support set of images. We provide different support sets of different sizes, and you should report results obtained with the different support sizes, as well as with different $k$ for your classifier. In your report, reflect on the challenges of zero-shot classification and how results vary depending on $k$ and the support size.

## 3.9 Task D6 (optional): Visualising embeddings

At this point you will have several versions of the backbone model $f$ trained in different ways: classification (tasks D1-D3) and the triplet loss (task D4). Each training scheme learns a different representation, and the goal of this task is to compare the representations in a visual way. Specifically, you will use t-SNE [4] to plot the $f(x)$ embeddings produced by your model $f$ trained from tasks D2 and D4 (the coarse-grained version), for all images in the test set. For the scatter plots, you will need to use the coarse-grained label to assign a colour to each embedding projection. You should include your visualisations in your report, and discuss how the embeddings cluster in different ways depending on the training scheme of the model. If you tried with different versions of $f$ from task D4 trained with different margin values for the triplet loss, how did the margin affect the visualisation? Similarly, if you tried with different triplet sampling scheme, how did this affect the embeddings?

### 3.10 Task D7 (optional): Checking generalisation

In this task you will test how well your model generalises. In particular, this task asks the question: does the model trained for *fine-grained* classification implicitly cluster images belonging to the same *coarse-grained* class? To check this you will need to implement a $k$-nn classifier where given a test image you predict the *coarse-grained* class based on the $k$ neighbours in the training set. Specifically, you will need to load the backbone model trained from task D1 (which was trained for fine-grained classification), extract embeddings for both the training and testing sets (without retraining the model), and run the $k$-nn classifier using the coarse-grained labels. The classifier will be evaluated with top-1 accuracy on the usual test set. You need to compare the results obtained with your $k-$nn classifier to those obtained with the classifier you trained in task D2, which was developed for coarse-grained classification. Are the two classifiers comparable, i.e. does the model trained on fine-grained classification implicitly learn the coarse-grained classes? Discuss this in your report, reflecting also on the challenges of generalisation and discussing how the performance of the $k-$nn classifier varies with $k$.

## 4 Criteria for Marking

| Task | Mandatory? | Max mark |
|---|---|---|
| C1 | Y | 6 |
| C2 | Y | 8 |
| C3 | Y | 8 |
| D1 | Y | 8 |
| D2 | Y | 8 |
| D3 | Y | 10 |
| D4 | Y | 20 |
| D5 | N | 12 |
| D6 | N | 8 |
| D7 | N | 12 |
| **Max mark with only mandatory tasks** | | 68 |
| **Max mark with mandatory tasks and at least one optional task** | | 76 |
| **Max mark with all tasks** | | 100 |

Table 3: Maximum mark for each task.

The maximum mark per task is detailed in Table 3. There is no penalty for not implementing a mandatory task - you will simply get 0 on the tasks that you skip. The marking criteria are explained in Table 4. Note that the report is the most important part of your assessment. We will run your code and check your output on a private test set with the exact same structure of the public test set. The performance of your output will influence your final mark, but we will not mark the quality of your code.

We will publish our results on the public test set so that you can gauge whether your models achieve decent results. Your results must pass a minimum threshold for each task, but there are no extra marks for any extra points your model scores compared to our implementation. In fact, please bear in mind that we care about the explanation of what you have done and the interpretation of the results more than the number themselves. This applies especially for tasks D1-D7: we know that getting good results with a CNN requires extensive training and fine-tuning. This is why we have chosen a small CNN and prepared a small subsample of the dataset to make training on an average CPU bearable (nevertheless, we recommend using Colab with GPU acceleration to speed things up). Below you can find further details about what we expect to see in a good report for each task:

- **Tasks C1-C3**. The methodology should be well-explained, incorporating relevant theories and technical details. Each step of the process should be clearly described, including the rationale behind the chosen techniques/functions. If the method involves multiple stages, present the results of each stage individually and discuss their contribution to the overall performance (if possible). You are expected to evaluate the effectiveness of your approach using suitable performance metrics. Consider whether the results can be improved by experimenting with alternative functions or adjusting key parameters. Discuss the sensitivity of your method to these parameters. How do changes affect the outcomes? Where applicable, provide visualisations to support your analysis and illustrate the impact of different configurations.
- **Tasks D1-D3**. An explanation of the architecture choice of your classifier and how your choices affected the performance of the model. For example, did having few or many layers result in a better performance? Did the hidden dimension of the linear layers affect performance? How about Dropout? What loss function and optimiser did you use, and why? If you tried more than one, did they differ in performance? How did hyper-parameters (e.g. learning rate, batch size) affect your results? Comparing tasks D1, D2, and D3, which one was easier/harder, which one got the best/poorest results, and why? Did you use the same classifier architecture for all three tasks?
- **Task D3**. How does the performance of the multi-class classifier compare to the performance of the single-class models you developed in tasks D1 and D2?
- **Task D4**. What triplets sampling strategy did you use and why? What metric did you use to calculate distance between embeddings, and why? Did you tune the margin hyper-parameter? How did performance vary according to this?
- **Task D5**. How do $k$ and the distance metric affect the performance of your zero-shot classifier? How do results vary depending on the support size? If you tried changing the architecture of the classifier, what changes did you make and why? Did they improve the performance compared to the design from D1?
- **Task D6**. How do the projected embeddings differ when the model is trained with different loss functions? Can you explain the differences? Did you try with different versions of the task D4 model trained with different margin values?
- **Task D7**. How do $k$ and the distance metric affect the performance? How good is your fine-grained model at predicting coarse-grained labels, despite not being trained for it?

| Criterion | Description |
|---|---|
| **Explanation of approach** | The method is explained well, and it clearly details how the various steps are carried out. |
| **Analysis of performance** | The performance is analysed and evaluated well with both quantitative and qualitative evaluation. |
| **Analysis of parameters** | The sensitivity of the task with respect to relevant parameters associated with the algorithm are analysed. |
| **Code output** | The code runs without errors and achieves good results on a held-out test set. Results will be compared with our own implementation. |

Table 4: Marking criteria.

Can you explain why is it good or bad?

# 5 Usage of library functions and restrictions

For any library function you use, please ensure you carefully read and understand its documentation. Note that some functions may behave differently depending on the operating system. Specific restrictions and permissions for each tasks are detailed below. Using external libraries other than the specified ones will cause issues when testing your code from our side.

## 5.1 Classic computer vision methods (C)

- Task C1: You may use library functions such as OpenCV for loading and saving images. However, functions that directly perform the task, such as `cv2.Canny()` and `cv2.HoughLines()` are not allowed.
- Task C1: Preprocessing functions like `cv2.Sobel()` and `cv2.filter2D()` are permitted.
- Tasks C2, C3: You may use library functions to create pyramids (e.g. `cv2.PyrUp`, `cv2.PyrDown`) or apply Gaussian convolution.
- Tasks C2, C3: You may use library functions for feature extraction (e.g. extracting SIFT features), as well as mathematical libraries (e.g. Singular Value Decomposition for computing the homography).
- Tasks C2, C3: The following are not allowed: 1) `cv2.matchTemplate` 2) `cv2.BFMatcher` 3) Any other automatic feature-matching functions.
- Tasks C2, C3: All feature-matching and the RANSAC algorithm must be implemented manually.

## 5.2 Deep-learning-based classification (D)

- Tasks D1-7: You can only use PyTorch, NumPy and Matplotlib. Any other library is not allowed (except for D6).
- Tasks D1-7: You cannot use the `torchvision.datasets.CIFAR100` class since we will use a custom subsample of the dataset. You will need to implement the dataset class yourself.
- Tasks D1-7: You can only use the model we prepared (with the specified pre-training weights).
- Tasks D1-3: You must implement the classifiers yourself using classes from the `torch.nn` module. You can use existing loss functions to train your model.
- Task D4: You cannot use any existing triplet sampling and loss functions, you must implement the triplet sampling and the triplet loss by yourself.
- Task D6: You can use any available t-SNE implementation, e.g. this one from scikit-learn.
- Task D5 and D7: You must implement the $k$-nn classifier yourself using PyTorch or NumPy.

# 6 Feedback and Getting Help

**Formative feedback** on your work will be offered throughout the duration of the coursework:

- During your labs/tutorials, the tutors will be available to answer questions and offer guidance. Please note that tutors will not be able to make decisions on your behalf regarding your assignment. They are there to discuss your ideas and offer advice.
- Use Moodle forums to post general questions or questions specific to your assignment. The unit teaching team will respond to these, as well as your peers. This way, we will create a repository of knowledge that will be available to all.

You will receive **summative feedback** on your work within 3 semester weeks of the submission deadline. The feedback will discuss your performance based on the criteria for marking, including what you did well and how specific components/sections could have been improved.

# 7 Academic Integrity

Your work will be checked to ensure that you have not plagiarised. For more information about the plagiarism policy at the University, see https://library.bath.ac.uk/referencing/plagiarism.

Remember that the published work that you refer to in your report should be clearly referenced in your text and listed in a bibliography section given at the end of your report. For more information, see https://library.bath.ac.uk/referencing/new-to-referencing.

Any code submitted will also be checked.

This is a **Type A** coursework. You cannot use Generative AI (GenAI) tools when completing this assessment. For more information, see this link.

# 8 FAQs

The FAQs will be provided on the unit's Moodle page. Note that we will keep updating this page with new questions and answers throughout the academic year.

# References


[1] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[2] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *International Conference on Computer Vision*, 2019.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[4] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 2008.