Politecnico di Milano

School of Industrial and Information Engineering
Master of Science Degree in Computer Science and Engineering
Department of Electronics, Information and Bioengineering

Semantic-aware Sampling for Robust Multi-model Fitting

Supervisor	Prof. Luca Magri
Co-supervisors	Prof. Giacomo Boracchi
	Dott. Filippo Leveni

Master graduation thesis of
Rizzo Antonino Maria
Student ID 898383

Academic Year 2019-2020

Ai miei genitori
ed Eleonora

“If you’re going to try,
go all the way.
Otherwise, don’t even start.”

Charles Bukowski
Roll the dice

Abstract

Robust multi-model fitting is the task of finding a series models that best fit a set of data corrupted by noise and outliers. This is an ubiquitous problem in *Computer Vision* where organizing unstructured visual data points in high level geometric structures is a necessary and basic step to derive better descriptions and understanding of a scene.

The sampling phase is a crucial step in robust multi-model fitting, therefore, in this work, we present a novel sampling strategy, termed *semantic-aware sampling*. In *Computer Vision*, visual data comes from pictures or frames of a video sequence, but state-of-the-art robust estimators are agnostic about the visual semantics of the points, they just treat visual data as geometric locations in an abstract space. On the contrary, we exploit the information that input points have been extracted from one or multiple pictures. This enhancement of the sampling process improves the performance of robust estimators while reducing the number of required iterations.

We propose to analyse the images by combining two approaches: a hand-crafted approach, where we extract a set of corresponding points, and a data-driven approach, where we obtain a probability map, termed *semantics*, that guides the sampling toward promising regions containing foreground objects rather than background.

Experiments show that this simple yet powerful approach significantly reduces the error of state-of-the-art robust estimators, thus improving model estimation.

Sommario

Il *robust multi-model fitting* consiste nel trovare i modelli che meglio descrivono un insieme di punti corrotti da rumore e valori anomali. Questo è un problema onnipresente nella *Computer Vision* in cui organizzare i punti di un'immagine secondo strutture geometriche di alto livello è un passaggio fondamentale per descrivere e comprendere meglio l'immagine.

La fase di campionamento è un passaggio fondamentale nella stima robusta dei modelli, per tale motivo, in questo elaborato, viene presentata una nuova strategia di campionamento, denominata *semantic-aware sampling.* In *Computer Vision*, i dati visuali provengono da immagini o frame di un video, ma gli stimatori robusti presenti nello stato dell'arte sono agnostici rispetto alla semantica dei punti, considerano solo la loro posizione su uno spazio astratto. Al contrario, il semantic-aware sampling sfrutta l'informazione che i punti sono stati estratti da una o più immagini. Migliorare il campionamento accresce le prestazioni degli stimatori robusti e riduce il numero di iterazioni richieste.

Si propone di analizzare le immagini combinando due approcci: un approccio *hand-crafted*, in cui viene estratto un insieme di corrispondenze tra i punti delle immagini, e un approccio *data-driven*, in cui si ottiene una mappa di probabilità, denominata *semantica*, che guida il campionamento verso regioni promettenti che contengano gli oggetti di interesse piuttosto che lo sfondo.

Gli esperimenti dimostrano che questo semplice ma potente metodo riduce significativamente l'errore degli stimatori presenti nello stato dell'arte migliorando così la stima dei modelli.

Ringraziamenti

Vorrei esprimere la mia gratitudine alle persone che mi sono state vicine in questo percorso. Inizierei ringraziando *Giacomo*, *Luca* e *Filippo* per avermi accolto nel loro gruppo di lavoro, per il supporto tecnico e per aver creduto in me anche quando io stesso ho dubitato. Grazie per le chiaccierate “pourparler”, mi hanno dato molto su cui riflettere.
Vorrei ringraziare anche *Mirko* e *Andrea*, non credo potrò mai ripagarvi per l’aiuto, i consigli e i bei momenti che mi avete regalato. Grazie a voi, ricorderò col sorriso questi anni.
Grazie di cuore ai miei genitori, *Lia* e *Calogero*, per i vostri sacrifici, per il vostro supporto incondizionato, per l’amore e l’affetto che mi avete trasmesso.
Grazie *Eleonora* per essermi sempre stata affianco, per avermi spronato a dare il massimo e per essere la mia fonte inesauribile di ispirazione.

Grazie a tutti voi.

Milano, 28 Aprile 2020

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Problem formulation

In this work, we address the *robust multi-model fitting* problem. First of all, let's describe the rationale behind the terminology. The word "fitting" means to find a mathematical function, or model, that best fits a set of data points, possibly corrupted by noise. For example, Fig. 1.1a shows a set of points scattered on an unknown straight line. We can apply the *ordinary least squares* (OLS) method to find the best line that approximate the data points by minimizing the sum of the squared residuals, where the residual of point i is computed as $|\tilde{y}(x_i) - y(x_i)|$, being $\tilde{y}$ the true model and y the estimation[1]. OLS returns the function $y = x$, colored green, which corresponds to the true model the points come from. Secondly, the word "robust" means that the methods work even in the presence of anomalous data points, called outliers. Let's go back to the OLS example. Fig. 1.1b shows that even adding a single outlier, OLS returns a very different function than the truth one and that is why we say OLS is sensitive to outliers. In the example, OLS returns the function $y = 0.3836x - 0.9863$, colored red, while the truth model is $\tilde{y} = x$, colored green and dashed. We are looking for methods that can automatically detect and ignore outliers during the estimation. We refer to such methods as "robust". Finally, the term "multi-model" means that we can have multiple functions describing the data points. Therefore, given a model, we have *gross-outliers*, i.e. points that cannot be explained by any model, and *pseudo-outliers*, i.e. points that can be explained by another model. Note, we say a model "explains" a point if the point lies closer than a threshold ε to the model. Summing up, we look for methods, called *robust estimators*, that

[1]OLS uses a residual measured only on the y-axis, the robust estimators that we will see later use a geometric residual, i.e., a value proportional to the point-model distance.

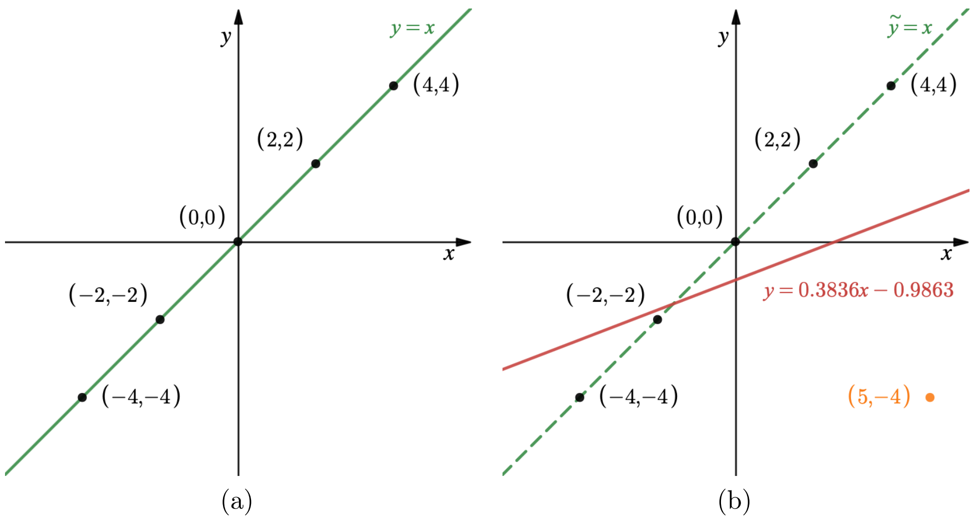


Fig. 1.1: Simple Ordinary Least Squares (OLS) example. (a) shows OLS result in case there are no outliers. The estimated function coincides with the true one, i.e. $y = x$. (b) shows OLS result in case there is an outlier, colored orange. The estimated functions is $y = \frac{1}{2}x - 1$, which is very different from the true one, i.e., $y = x$. Even a single outlier can harm the OLS estimation.

can detect multiple models that describe the data points in the presence of outliers and noise.

Robust multi-model fitting is an ubiquitous problem that can be encountered in many *Computer Vision* applications. For example, it is used in the *template matching* task to aggregate point-correspondences belonging to the same template discarding wrong matches produced because of background noise and clutter. Also, it is used in 3D reconstruction to estimate multiple rigid moving objects so to initialize the multi-body Structure-from-Motion pose graph [11][24]. Other scenarios where the estimation of multiple models plays a primary role include face clustering, pose estimation and video motion segmentation, just to name a few.

1.2 The sampling phase

Most popular robust estimators implement a sampling phase: they randomly sample a minimal set of points, called *minimal sample set* (MSS), required to uniquely determine the free parameters of the model. For example, two points are required to fit a line, three points are required to fit a circle, four points are required to fit a *homography* and eight points are required to fit a

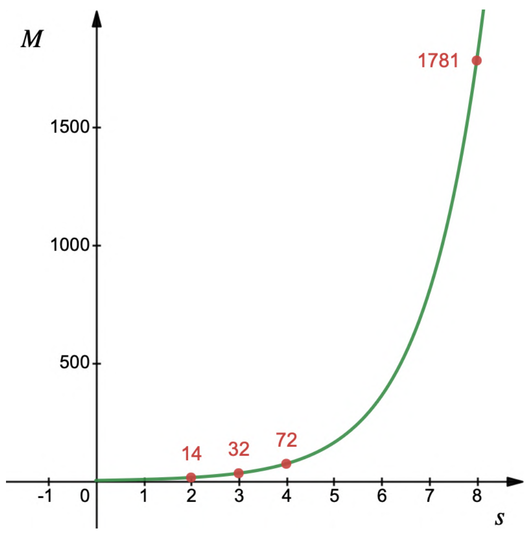


Fig. 1.2: Probabilistic estimate for the number of samplings M given the cardinality of the MSS, denoted as s. The plot depicts the case where the probability of success is 0.95 and the inlier ratio is 0.45. For $s = 2$ we need 14 samplings, for $s = 3$ we need 32 samplings, for $s = 4$ we need 72 samplings, for $s = 8$ we need 1781 samplings. It is an exponential growth.

fundamental matrix[2]. The rationale is that by using a few points it is more likely to select no outliers and fitting the correct model. This simple, yet powerful, idea allows the methods to be robust to outliers. Note, this is the opposite of OLS which is data hungry and the more data it has, the better the model it can estimate. Having a sampling phase has a drawback: to make sure we find the models that better explain the data, we should sample all possible MSSs. This would take too long, so the common practice is to calculate a probabilistic estimate. However, we have only shifted the problem: Fig. 1.1b shows that, as the cardinality s of the MSS increases, the number of required samplings M becomes too expensive.

The number of samplings can be significantly reduced if we use a more sophisticated sampling strategy than uniform sampling. A popular choice is the use of localized sampling [15], i.e., we select neighboring points in the ambient space as they are more likely to be part of the same geometric model. This strategy helps us to limit the number of required samplings but, since different models can obey different spatial distributions, it could also be harmful.

[2]The homography and the fundamental matrix will be described in Sec. 4.1.3.2 and Sec. 4.1.3.6, respectively.

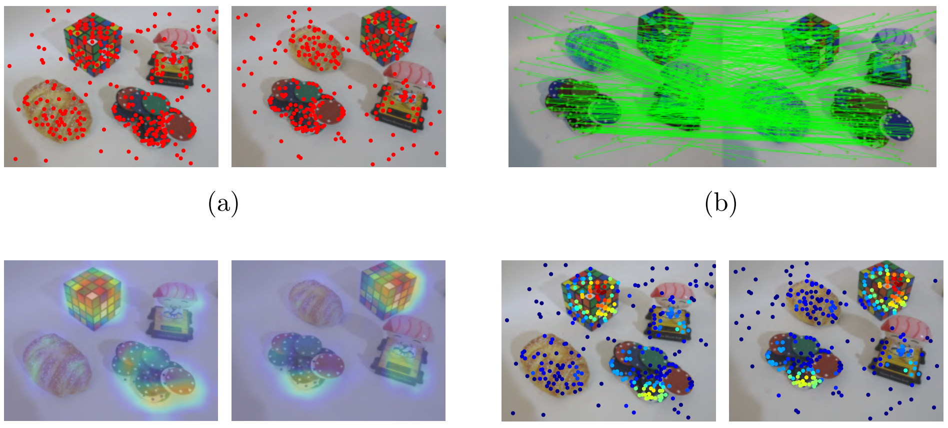


Fig. 1.3: Embed semantics into discrete probability. (a) shows the points over the picture. (b) shows the point matches. (c) show the CAM M_c with $c =$ "bread". (d) graphically represents the discrete probability vector p. Original figure from AdelaideRMF data-set [25].

1.3 Our contribution

As described in Sec. 1.1, robust estimation is the solution to many Computer Vision tasks where the sets of input points have been extracted from pictures. As far as we know, state-of-the-art robust estimators just ignore the picture, only consider the location of the points, and are completely unaware of their semantics. We believe that exploiting point semantics can improve the performance of robust estimators. We propose to extract the semantics of the picture using a *Convolutional Neural Network* and embed this information into the estimator by implementing a sophisticated sampling strategy that we call *semantic-aware sampling*. An example is shown in Fig. 1.3. Given an image pair, we independently extract the keypoints from each image, as shown in Fig. 1.3a and match the left and right image-points according to their local description, as shown in Fig. 1.3b. Moreover, for each picture we get its semantics by extracting a probabilistic map, as shown in Fig. 1.3c. Finally, we assign this probability to the matches to obtain a discrete probability vector as graphically shown in Fig. 1.3d and use it in the sampling phase along with locality information. In the example, the semantics we extracted from the image focuses on the cube, but we will extract many probabilistic maps, each focused on a different part of the pictures, so we will also have maps that highlight the other objects depicted in the image, namely bread, toy and chips.

The advantage of our method is twofold: it allows to sample many more pure MSSs, i.e., MSSs with neither gross-outliers nor pseudo-outliers, thus reducing the number of required samplings and, when embedded into robust estimation, it allows to achieve a lower misclassification error than state-of-the-art methods.

1.4 Outline

The remaining of this work is structured as follows:

- Chapter 2 introduce to the concepts of *preference* and *consensus* and presents some popular robust estimators for each approach.
- Chapter 3 deals with the importance of good sampling and describes different sampling strategies for the multi-model scenario.
- Chapter 4 presents the two approaches to image processing: the hand-crafted approach, where we explicitly design the feature extractor, and the data-driven approach, where we let the model learn the features by itself.
- Chapter 5 describes our proposed method by providing both intuition and formal description along with the pseudo-code.
- Chapter 6 discusses the experimental setup and the comparison with the state-of-the-art methods.
- Chapter 7 is devoted to conclusions and future work.

Chapter 2

Robust model fitting

As mentioned in the introduction, *robust model fitting* is the problem of finding a model, or multiple models, explaining the data in the presence of noise and outliers. The algorithms tackling this problem are called *robust estimators* and we devote this chapter to their presentation.

2.1 Consensus and preferences

In robust model fitting, we can recognize two broad families of estimators: *consensus*-based and *preference*-based. In order to define these two approaches, it is necessary to introduce the notion of *residual*. A residual is a measure of how well the model $\theta \in \Theta$ fits a point $\mathbf{x} \in X$ and can be formally described as a function:

$$r : X \times \Theta \to \mathbb{R}^+ \tag{2.1}$$

that associate to each point-model pair $(\mathbf{x}, \theta) \in X \times \Theta$ a positive value proportional to the point-model distance. Given a model θ, points that have residual less than a certain threshold ε are called *inliers* while points that have residual greater than ε are called *outliers*.

The *consensus set* of a model is the set of its inliers. Consensus-based methods select the model(s) with the largest consensus or, equivalently, with the largest number of inliers. The most representative algorithms of this family are RANSAC (Sec. 2.2) and its variants to the multi-model case (Sec. 2.3).

The *preference set* of a point is the set of models having that point as inlier. Preference-based methods exploit the *preference trick*: points of the same true model are inliers for the same estimated models. Thus, we can use the estimated models to represent the points in a new conceptual space, called *preference space*, where inliers to the same true model form groups and can be clustered. Methods that follow the preference approach implement a two

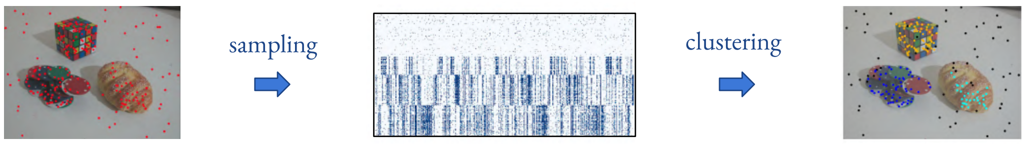


Fig. 2.1: First-represent-then-clusterize scheme. At first, we sample a bunch of models and we build a matrix that we will use to cluster the points in the preference space. This matrix is called *preference matrix* and is shown in the middle as a white-blue box where blue dots are 1 and white dots are 0. Original figure from AdelaideRMF data-set [25].

steps *first-represent-then-clusterize* scheme, as illustrated in Fig. 2.1. Given a set of N data points, a bunch of M models is randomly sampled in order to build a matrix Ω called *preference matrix* with N rows and M columns. The columns of the matrix represent the consensus set of each model while the rows represent the preferences. We will use the rows of the preference matrix to describe the points in the preference space. The most representative algorithms in this family are J-Linkage (Sec. 2.4) and T-Linkage (Sec. 2.5).

2.2 RANSAC

The Random Sample Consensus (RANSAC) algorithm [1] is a simple yet powerful method for estimating the parameters of a model in the presence of noisy data and outliers. Given a set of N data points, RANSAC iterates M times three operations:

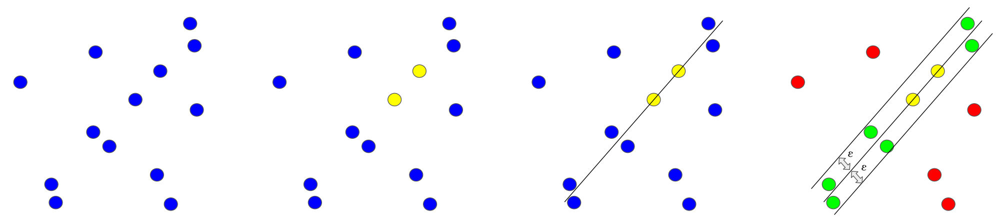
i) randomly samples a *minimal sample set* (MSS), i.e., set of points required to uniquely fit a model;

ii) uses the points in MSS to fit a model θ_m;

iii) counts the inliers of θ_m.

At the end RANSAC returns the model with the largest number of inliers or, equivalently, with the largest consensus. Fig. 2.2 show an iteration of RANSAC. Note, the model type must be known in advance.

The cardinality of the MSS corresponds the minimum number of points required to uniquely compute the parameters of the model θ_m. For instance, a straight line needs two points to be uniquely determined, a circle needs three points, a homography needs four points and a fundamental matrix needs eight points. The homography and the fundamental matrix will be described in Sec. 4.1.3.2 and Sec. 4.1.3.6, respectively. RANSAC is formally described in Alg. 1. As a final remark, note that the inlier threshold ε is a critical parameter because the sparsity of the inliers is not always the same.



(a) Set of points. (b) Sample a MSS. (c) Fit θ_m (line). (d) Count θ_m inliers.

Fig. 2.2: Example iteration of RANSAC. Two points are randomly sampled to fit a straight line and its inliers are computed. Yellow points represent the MSS, green points the inliers to θ_m while red points are the outliers. ε is the inlier threshold.

Algorithm 1: RANSAC

```
Input  : {x_n | n = 1, ..., N} – set of data points
         s – number of samples required to fit the model
         ε – inlier threshold
         M – number of iterations
Output : θ* – best model
         n* – number of inliers of θ*

n* ← 0
for m = 1, ..., M do
    S ← Draw a minimal sample set of s points
    θ_m ← Fit the model using S
    n ← Count the inliers of θ_m using ε as inlier threshold
    if n > n* then
        n* ← n
        θ* ← θ_m
    end
end
return θ*, n*
```

2.3 Sequential RANSAC & Multi-RANSAC

To make RANSAC estimate multiple structures we can repeat its execution many times. At each iteration, RANSAC finds the model θ_m with the largest consensus, stores it and removes its inliers. We stop when there are no models with sufficiently large consensus. This extension of RANSAC to the

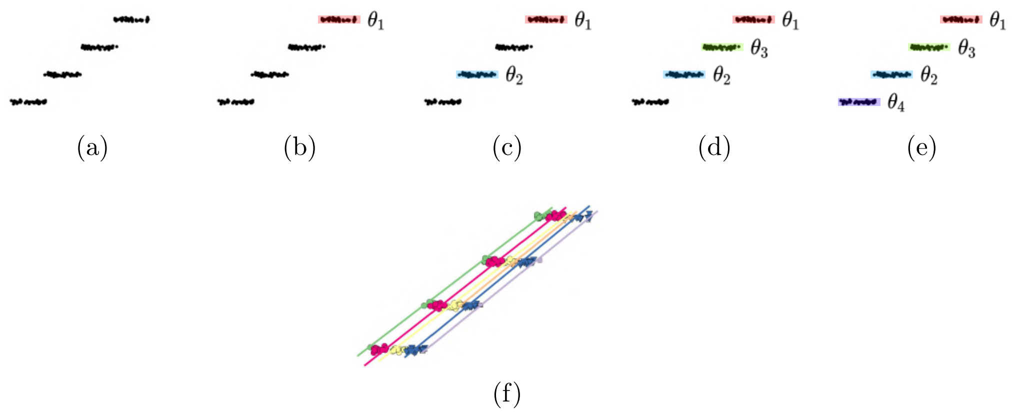


Fig. 2.3: Sequential RANSAC. Figure (a) shows a set of points, figures (b) to (e) show four iterations of Sequential RANSAC that produce good model estimations. At each iteration we find a model θ_i and remove the inliers with respect to θ_i to run RANSAC again. The main limitation comes from its sequential behavior. If one of the first estimates is poor, we will remove points that are inliers to other models and this will lead us to bad results. An example is shown in (f). Original figure from [17].

multi-model case is called Sequential RANSAC. An example execution is shown in Fig. 2.3 (from Fig. 2.3a to Fig. 2.3e). As pointed out by [17], the major drawback of Sequential RANSAC is its greediness: if one of the initial structures is inaccurate, the overall result is poor. An example of this scenario is shown in Fig. 2.3f where the models we obtain are not the desired ones.

Multi-RANSAC [17] avoids the sequential behavior of Sequential RANSAC, which is at the root of its issues, and simultaneously estimates k models. At each iteration, the k models are updated with k new sample models using a fusion procedure that explicitly enforces the disjointness of the consensus set. However, as demonstrated in [19], the method has poor results in the case of intersecting structures.

2.4 J-Linkage

J-Linkage [19] is a method explicitly designed for robust multi-model fitting that follows the preference approach. As described in Sec. 2.1, we randomly sample a pool of M models and we build the preference matrix Ω of size $N \times M$, where N is the number of data points. For each (i, j) with $i \in \{1, \dots, N\}$ and $j \in \{1, \dots, M\}$:

$$\Omega(i,j) = \begin{cases} 1 & \text{if} r(i,j) \leq \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

As mentioned in Sec. 2.1, the rows of Ω are the *preference sets* $PS \in \{0,1\}^M$ of the points. An example preference matrix is shown in Fig. 2.4a. Once we have computed Ω, we have the description of the points in the preference space. In addition, we need a concept of distance in this space. J-Linkage uses the *Jaccard distance* that measures the degree of overlap of two sets:

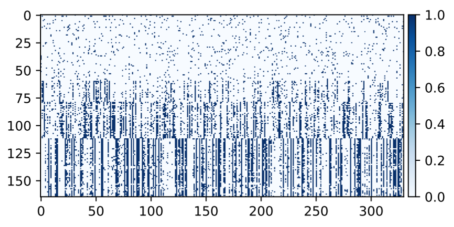
$$\mathrm{d_J}(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{2.3}$$

where A and B are two preference sets (rows of Ω). Note, $\mathrm{d_J}$ ranges from 0 (identical sets) to 1 (disjoint sets).

J-Linkage is described in Alg. 2. It proceeds in a bottom-up manner: starts from singletons and at each iteration merges the two clusters with the smallest Jaccard distance. The preference set of the just merged cluster is computed as the intersection of the preference sets of its points. The algorithm ends when the distance between the two closest clusters is 1, which means that the algorithm merge clusters whose preference sets overlap, i.e., have at least a 1 in common. Finally, we can estimate a model from each cluster by least square fitting. Note, the choice of the inlier threshold is critical as in RANSAC.

Algorithm 2: J-Linkage

```
Input  : set of data points represented by their preference set.
Output : clusters of points belonging to the same model.

1  Put each point in its own cluster.
2  while True do
3      Compute the Jaccard distance between all the clusters.
4      Find the two cluster $C_i$ and $C_j$ with the smallest distance $d_{min}$.
5      if $d_{min} = 1$ then
6          End clustering.
7      end
8      if $d_{min} < 1$ then
9          Merge $C_i$ and $C_j$ in a new cluster $C$.
10         Compute $\mathrm{PS}_C$ as $\mathrm{PS}_{C_i} \cap \mathrm{PS}_{C_j}$.
11     end
12 end
```


(a)

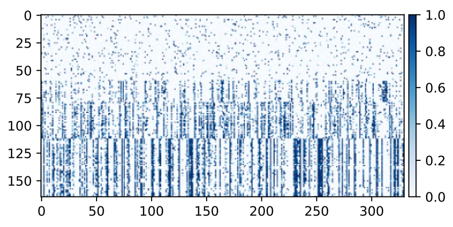

(b)

Fig. 2.4: Comparison between the preference matrix of J-Linkage and T-Linkage. (a) shows the preference matrix for J-Linkage; (b) shows the preference matrix for T-Linkage.

	J-Linkage	T-Linkage
Space	$\{0,1\}^M$	$[0,1]^M$
Cluster	$\bigcap PS$	$\min PF$
Similarity	Jaccard	Tanimoto

Table 2.1: Comparison J-Linkage and T-Linkage.

2.5 T-Linkage

A critical parameter of RANSAC (Sec. 2.2) and J-Linkage (Sec. 2.4) is the inlier threshold ε: points that lie immediately before the threshold are inliers, while points immediately after are outliers. T-Linkage [27] improves J-Linkage by relaxing this rigid behavior by assigning a continuous value in $[0,1]$ to the points rather than a binary value in $\{0,1\}$ (outliers or inliers). Given a set of N points and M sampled models, for each (i,j) with $i \in \{1,\dots,N\}$ and $j \in \{1,\dots,M\}$ we define the (i,j) entry of the preference matrix as:

$$\Omega(i,j) = \begin{cases} e^{-\frac{r(i,j)}{\tau}} & \text{if } r(i,j) \leq 5\tau \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where τ plays the same role of the inlier threshold ε, but it is less critical since it replaces the abrupt truncation of Eq. 2.2 with an exponential decay. Points are represented by their preference function $PF \in [0,1]^M$, which is an extension to the continuous space of the preference set $PS \in \{0,1\}^M$. An example preference matrix is shown in Fig. 2.4b. Moreover, the Jaccard distance is replaced with the *Tanimoto distance*

$$\mathrm{d_T}(A,B) = 1 - \frac{<p,q>}{\|p\|^2 + \|q\|^2 - <p,q>} \tag{2.5}$$

where p and q are two preference functions (rows of Ω), $< \cdot, \cdot >$ indicates the inner product and $\|\cdot\|$ indicates the norm. The clustering method is described in Alg. 3. As for J-Linkage, T-Linkage starts from singletons and at each iterations merge the two cluster with lower distance. The preference function of the resulting cluster is computed as the element-wise minimum. A comparison of J-Linkage and T-Linkage is reported in Tab. 2.1. Finally, we can estimate a model from each cluster by least square fitting.

Algorithm 3: T-Linkage

```
Input  : set of data points represented by their preference function.
Output : clusters of points belonging to the same model.

1  Put each point in its own cluster.
2  while True do
3      Compute the Tanimoto distance between all the clusters.
4      Find the two cluster $C_i$ and $C_j$ with the smallest distance $d_{min}$.
5      if $d_{min} = 1$ then
6          End clustering.
7      end
8      if $d_{min} < 1$ then
9          Merge $C_i$ and $C_j$ in a new cluster $C$.
10         Compute $\mathrm{PF}_C$ as the component-wise min of $\mathrm{PS}_{C_i}$ and $\mathrm{PS}_{C_j}$.
11     end
12 end
```

Chapter 3

Sampling

In the description of RANSAC(Sec. 2.2), J-Linkage (Sec. 2.4) and T-Linkage (Sec. 2.5) we have defined a parameter M which corresponds to the overall number of uniformly sampled models. While RANSAC only keeps the model with the largest consensus, J-Linkage and T-Linkage use all models to move to the preferences space. So far we have not specified the value of M because it depends on the sampling strategy. In this chapter we will see that the better the sampling the lower the value of M we can choose.

3.1 The importance of good sampling

In order to define the number of samplings, it is necessary to introduce a measure of goodness for the samplings. In the context of robust multi-model fitting, estimators are designed to deal both with *gross-outliers*, i.e., points not belonging to any model and *pseudo-outliers*, i.e., points belonging to other models. To this end, they do not consider all the points at once, but they only sample a minimal sample set (MSS). When the MSS contains points all belonging to the same model, we denote it as *pure*. In other words, a pure MSS is a gross-outlier-free and pseudo-outlier-free MSS.

In the single-model case, since RANSAC (or one of its variants) just keep the model with the largest consensus, we are interested in sampling at least one pure MSS. If we wanted to be sure to sample a pure MSS at least once, we would have to try all possible combinations of points, i.e., $\binom{N}{s}$ samplings, where N represents the number of points and s is the cardinality of the MSS. For a large N this would take too long and that is why it is common to use a a probabilistic estimate as:

$$M = \frac{\ln(1-p)}{\ln\left(1-(1-e)^s\right)} \tag{3.1}$$

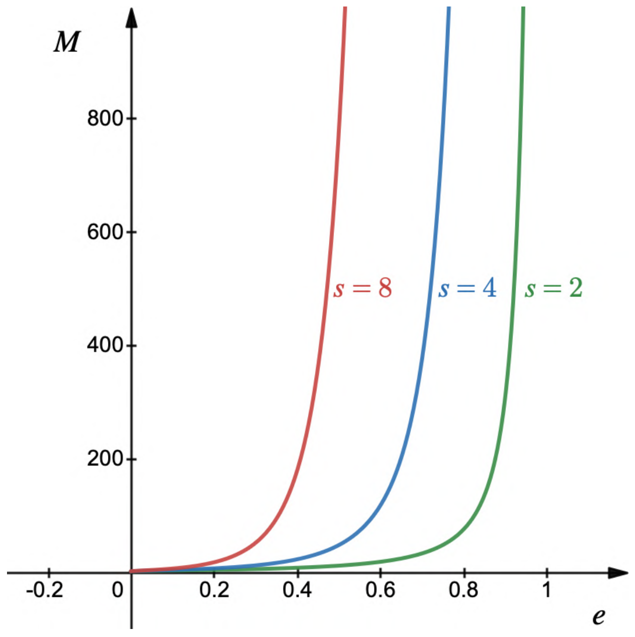


Fig. 3.1: Plot of the number of required samplings for different values of the outlier ratio e and the MSS cardinality s.

where p is the probability of finding the optimal solution within M uniform samplings and e is the outlier ratio. However, we have only shifted the problem: from Fig. 3.1 it can be seen that, with the same outlier ratio e, increasing the cardinality s of the MSS, exponentially increases the number M of samplings required to have a pure MSS with probability p. Note that Eq. 3.1 is an underestimation because it does not take noise into account. The estimate is even more severe in the multi-model setting and becomes crucial for methods that exploit the preference trick. When multiple models describe the data-points we need to sample at least a pure MSS for each model. Also, having a large number of pure MSS allows for a better description of the points in the preference space and makes it harder for greedy algorithms (like J-Linkage and T-Linkage) to cluster points from different geometric models.

The number of samplings can be significantly reduced if we use a more sophisticated sampling strategy than uniform sampling.

3.2 Localized Sampling

In this section, we present the *localized sampling* strategy [15] proposed to increase the probability of sampling points all belonging to the same geometric model. The rationale is that points belonging to the same model are likely to be close, therefore the idea is to sample neighboring points. In practice, we sample a first point α with uniform probability and compute the distance of every point $\beta \neq \alpha$ with respect to α as:

$$d_{\alpha\beta} = \sqrt{(x_\alpha - x_\beta)^2 + (y_\alpha - y_\beta)^2} \tag{3.2}$$

From $d_{\alpha\beta}$, the probability of point β having already sampled α is computed as:

$$p(\beta|\alpha) = \begin{cases} \frac{1}{Z_\alpha} e^{-s_\alpha d_{\alpha\beta}^2} & \alpha \neq \beta \\ 0 & \alpha = \beta \end{cases} \tag{3.3}$$

where

$$Z_\alpha = \sum_{\beta \neq \alpha} e^{-s_\alpha d_{\alpha\beta}^2} \tag{3.4}$$

and s_α is chosen heuristically.

3.3 Multi-GS

Multi-GS [23] is a guided sampling strategy driven only by residual sorting information. Given a set of N input data-points $\{\mathbf{x}_i \mid i = 1, \ldots, N\}$, M geometric models $\{\theta_j \mid j = 1, \ldots, M\}$ are randomly sampled. For each point $\mathbf{x}_i$ we compute its absolute residuals with respect to the M sampled models:

$$\mathbf{r}^{(i)} = \begin{bmatrix} r_1^{(i)} & r_2^{(i)} & \ldots & r_M^{(i)} \end{bmatrix} \tag{3.5}$$

We denote by a_k the index of the k-th geometric model. For point $\mathbf{x}_i$, we sort the model indices $a_k^{(i)}$ in non-descending order based on the corresponding residuals:

$$u \leq v \Longrightarrow \mathbf{r}^{(i)}_{a_u^{(i)}} \leq \mathbf{r}^{(i)}_{a_v^{(i)}} \tag{3.6}$$

In practice, we rank the models according to the preference of $\mathbf{x}_i$. The lower the residual, the more the model is preferred. Two points $\mathbf{x}_i$ and $\mathbf{x}_j$ will share many common models at the top of their preference lists if they are inliers of the same geometric model. Let $\mathbf{a}_{1:h}^{(i)}$ be the vector with the first h elements of $\mathbf{a}^{(i)}$. We define the "intersection" between $\mathbf{x}_i$ and $\mathbf{x}_j$ as:

$$f(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{h} \left| \mathbf{a}_{1:h}^{(i)} \cap \mathbf{a}_{1:h}^{(j)} \right| \tag{3.7}$$

where $\left| \mathbf{a}_{1:h}^{(i)} \cap \mathbf{a}_{1:h}^{(j)} \right|$ finds the number of common models shared by $\mathbf{a}_{1:h}^{(i)}$ and $\mathbf{a}_{1:h}^{(j)}$. Window size h specifies the number of leading models to take into account. We can choose h in the range $\lceil 0.05M \rceil \leq h \leq \lceil 0.4M \rceil$.

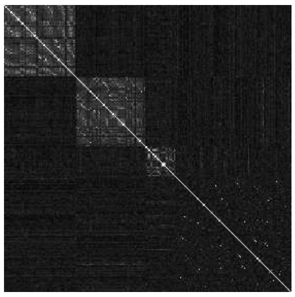


Fig. 3.2: $N \times N$ matrix K: element (i, j) is the number of common models shared between points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the first h positions. Figure from [23].

As shown in Fig. 3.2, we obtain a $N \times N$ matrix, denoted as K, where the element (i, j) is $f(\mathbf{x}_i, \mathbf{x}_j)$, i.e., the ratio of models shared between points $\mathbf{x}_i$ and $\mathbf{x}_j$. The rows of K show that an inlier concentrates mostly on inliers from the same geometric model, while for a gross-outlier the row values are generally low and appear to be randomly distributed. The idea of Multi-GS is to use the rows of K to drive the sampling phase. The first point $\mathbf{x}_i$ is selected from a discrete uniform distribution. To sample the second point $\mathbf{x}_j$, we look at the i-th row of K and we use this row as discrete probability over points to drive the sampling. To sample other points, we get the probability vector as the element-wise multiplication of the rows of the points previously sampled.

3.4 DGSAC

The Density Guided Sampling and Consensus (DGSAC) [39] is an automatic pipeline for robust multi-model fitting that exploits the concept of *Kernel Residual Density* (KRD) to differentiate between inliers and outliers. DGSAC begins with generating model hypotheses using density guided sampling (KDGS). Given a set of N data-points $\{\mathbf{x}_i \mid i = 1, \dots, N\}$, M geometric models $\{\theta_j \mid j = 1, \dots, M\}$ are randomly sampled. We denote by r_j^i the residual of point $\mathbf{x}_i$ with respect to $\mathbf{h}_j$. In the following, we introduce two vectors $\mathbf{q}_j$ and $\mathbf{l}^i$ which encode the preference of models over points and the preference of points over models, respectively. For each model $\mathbf{h}_j$, we sort point indices q_j^i according to the ascending order of the residuals:

$$\mathbf{q}_j = \begin{bmatrix} q_j^1 & q_j^2 & \dots & q_j^N \end{bmatrix} \qquad \text{s.t.} \qquad r_j^{q_j^1} \leq r_j^{q_j^2} \leq \dots \leq r_j^{q_j^N} \tag{3.8}$$

For each point $\mathbf{x}_i$, we sort model indices l^i_j according to the ascending order of the residuals:

$$\mathbf{l}^i = \begin{bmatrix} l^i_1 & l^i_2 & ... & l^i_M \end{bmatrix} \qquad \text{s.t.} \qquad r^i_{l^i_1} \leq r^i_{l^i_2} \leq ... \leq r^i_{l^i_M} \tag{3.9}$$

Since we use the residual in two different flavors, we refer $\mathbf{q}_j$ as *residual based hypothesis* and $\mathbf{l}^i$ as *point preferences.* In practice, $\mathbf{q}_j$ ranks first the points that lie closer to model $\mathbf{h}_j$, while $\mathbf{l}^i$ ranks first the models that better describe point $\mathbf{x}_i$.

Given a model $\mathbf{h}_j$ and a point $\mathbf{x}_i$, we define the *kernel residual density* (KRD) of point $\mathbf{x}_i$ with respect to model $\mathbf{h}_j$ as:

$$d^{q^i_j}_j = \frac{1}{N}\sum_{k=1}^{N} \frac{1}{b^{q^i_j}_j} \; K\left(\frac{r^{q^i_j}_j - r^{q^k_j}_j}{b^{q^i_j}_j} \right) \tag{3.10}$$

where we set the bandwidth $b^{q^i_j}_j$ for point $\mathbf{x}_{q^i_j}$ as $r^{q^i_j}_j$ and $K(\cdot)$ can be any symmetric kernel function. We use the *Epanechnikov kernel*

$$K(u) = \frac{3}{4}(1-u^2) \qquad \text{s.t.} \qquad |u| \leq 1 \tag{3.11}$$

but other kernels (like Gaussian) can also be used. For good hypotheses, KRD is large near the regression surface (inlier region) compared to the outlier region. For a bad hypothesis, KRD is nearly flat throughout.

We can exploit the KRD to compute the *KRD-based point preferences.* For each point $\mathbf{x}_i$, we sort model indices v^i_j according to the descending order of the kernel densities:

$$\mathbf{v}^i = \begin{bmatrix} v^i_1 & v^i_2 & ... & v^i_M \end{bmatrix} \qquad \text{s.t.} \qquad d^i_{v^i_1} \geq d^i_{v^i_2} \geq ... \geq d^i_{v^i_M} \tag{3.12}$$

The pairwise point correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$ can be computed as:

$$c^{ij} = \frac{\mathbf{v}^i_{1:T} \cap \mathbf{v}^j_{1:T}}{T} \tag{3.13}$$

where $T = 5$. Once we compute c^{ij} for all the point couples, we obtain a $N \times N$ matrix, denoted as C. In practice, we move into a density space where the distance between the points is computed with c^{ij}. An example of matrix C is shown in Fig. 3.3b.

We can use the KRD in the sampling phase to obtain the KRD-Guided Sampling (KDGS). The idea is to combine *point preferences* and *kernel densities.* For each minimal sample set (MSS) with cardinality η, we sample

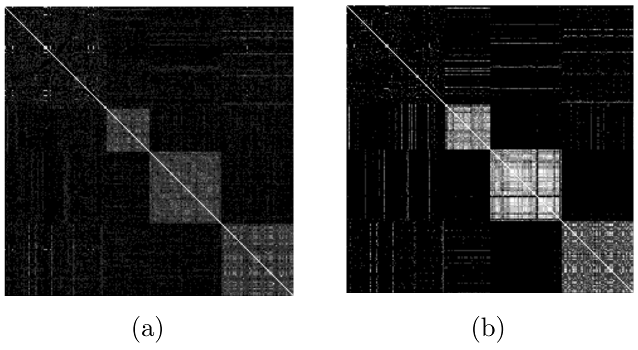


Fig. 3.3: The figure shows two $N \times N$ matrices, where N is the number of points. The element (i, j) represents the correlation between point $\mathbf{x}_i$ and $\mathbf{x}_j$. (a) show the residual-based point correlation; (b) show the density-based point correlation. Figure from [39].

the first point $\mathbf{x}_i$ deterministically and the remaining $\eta - 1$ points without replacement according to a discrete probability distribution that we describe below. For each point $\mathbf{x}_i$, we compute the potentially good model hypotheses. To this end, we take the *residual based hypothesis*: if the point appears within the first β positions, we select that model as good. In the set of potentially good models, we select the model h_{den} that maximizes the density with respect to $\mathbf{x}_i$ and the model h_{res} that minimizes the average residual for the first β points in the *residual-based hypothesis*. Thus, for each $\mathbf{x}_i$, we get two profiles (vectors): a *density profile* with the densities of all the points with respect to h_{den} and a *residual profile* with the residual of all the points with respect to h_{res}. We invert the residual profile to have high quantities for points we prefer. We normalize both profiles and compute the element-wise product. This way we computed a discrete probability matrix S that take into account the best model that can describe each point from a density and residual perspective. During sampling, if a point $\mathbf{x}_i$ was sampled, we consider i-th row of matrices C and S, we compute the element-wise product and we use the resulting vector as discrete probability distribution.

As mentioned, we sample the first point deterministically. We prepare a vector ν with all the point indices and at each iteration we consume an element of the vector. This way all the points are sampled at least once and KDGS stops when ν is empty. We can also add an *explanation score* that remove models that were already sampled many times during the random sampling.

Once all the models have been sampled, we run a *model selection algorithm*. Two strategies were proposed: a *greedy* strategy and an *optimal* one which is modeled as a quadratic problem. Being that in this work we are interested in

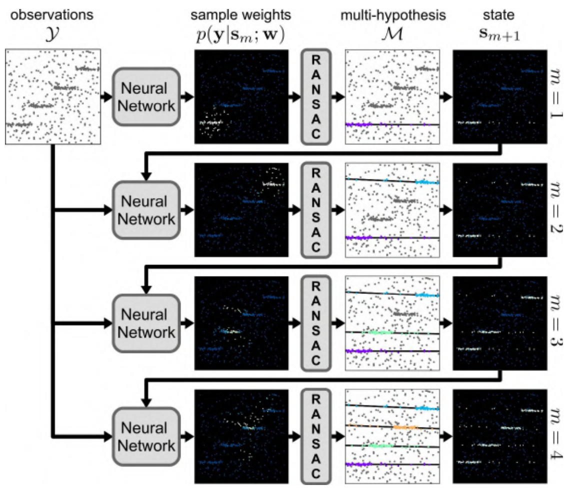


Fig. 3.4: CONSAC scheme. Figure from [40].

the sampling strategy, we will not go into the model selection part. For an exhaustive description, the reader is referred to [39].

3.5 CONSAC

CONSAC [40] extends Sequential RANSAC and is the first method that approaches the sampling strategy from a learning perspective. Suppose there are M instances of a geometric model $\mathbf{h}$ (such as line, homography or fundamental matrix) apparent in the data $\mathcal{Y}$. We denote the set of all the model instances as $\mathcal{M} = \{\mathbf{h}_1, ..., \mathbf{h}_M\}$. CONSAC estimates $\mathcal{M}$ via three nested loops:

1. **Single Model Instance Sampling.** It uses RANSAC to compute a hypothesis pool $\mathcal{H} = \{\mathbf{h}_1, ..., \mathbf{h}_T\}$ via random sampling of T MSS and selects the model $\hat{h}$ with the larger consensus. This level corresponds to one row of Fig. 3.4.

2. **Multi-Hypothesis Generation.** It repeats the single model instance sampling M times to fully generate $\mathcal{M}$. At each new iteration, a new model $\hat{\mathbf{h}}_m$ is selected so that it not only maximizes its consensus, but maximizes the joint consensus of all selected models up to the current iteration.
$$\hat{\mathbf{h}}_m = \underset{\mathbf{h}\in\mathcal{H}_m}{\arg\max}\; g_{\mathrm{s}}(\mathbf{h}, \mathcal{Y}, \hat{\mathbf{h}}_{1:(m-1)}) \tag{3.14}$$

where g_s measures the joint inlier count of the current hypothesis with respect to the previous ones. This level corresponds to the entirety of Fig. 3.4.

3. **Multi-Hypothesis Sampling.** CONSAC repeats steps 1 and 2 to independently generate a pool $\mathcal{P} = \{\mathcal{M}_1, ..., \mathcal{M}_P\}$ of candidate solutions. Finally, as shown in Eq. 2.2 , the solution with the larger joint consensus is selected.

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}\in\mathcal{P}} g_\mathrm{m}(\mathcal{M}, \mathcal{Y}) \tag{3.15}$$

where g_m measures the joint inlier count of all the hypotheses in $\mathcal{M}$.

Let's now discuss about the sampling strategy adopted by CONSAC. At each iteration, a neural network predicts the sample weights for each point $\mathbf{y}_i$ conditioned on a states $\mathbf{s}_m$ that encodes the information about the m previously sampled models. We define the state entry $s_{m,i}$ of observation $\mathbf{y}_i$ as:

$$s_{m,i} = \max_{j\in[1,m)} g_\mathrm{y}(\mathbf{y}_i, \hat{\mathbf{h}}_j) \tag{3.16}$$

where $i \in [0, |\mathcal{Y}|]$ and g_y measures if $\mathbf{y}_i$ is an inlier of model $\hat{\mathbf{h}}_j$. Note, $s_{m,i}$ measures if so far we have found a model describing point $\mathbf{x}_i$. A visualization of the state vector is shown in the last column of Fig. 3.4.

Before running CONSAC, we need to train the neural network in order to increase the chance of sampling pure MSSs. The network (shown in Fig. 3.5) can be trained end-to-end both in a supervised or self-supervised learning manner. In order to update the network weights $\mathbf{w}$, we approximate the gradients of the expected task loss $\ell(\hat{\mathcal{M}})$ by drawing K multi-model instances:

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}) \approx \frac{1}{K}\sum_{k=1}^{K}\left[\ell(\hat{\mathcal{M}}_k)\frac{\partial}{\partial \mathbf{w}}\log p(\mathcal{P}_k; \mathbf{w})\right] \tag{3.17}$$

where $\mathcal{P}$ is a pool of multi-model instances. If ground-truth models are available, we can use the task specific loss $\ell(\hat{\mathbf{h}}, \mathbf{h}^{\mathrm{gt}})$ minimizing the difference between the estimated model and the ground-truth one. In absence of ground-truth labels, we can train CONSAC in a self-supervised fashion by replacing the task loss with another quality measure, such as the joint inlier counts of the estimated models. For an exhaustive description, the reader is referred to [40].

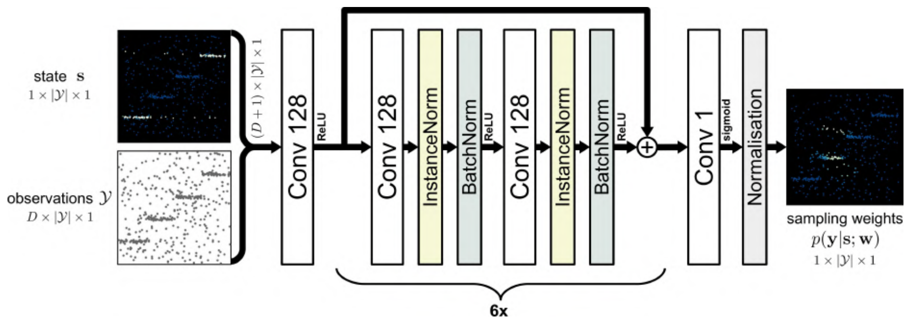

state $\mathbf{s}$
$1 \times |\mathcal{Y}| \times 1$
observations $\mathcal{Y}$
$D \times |\mathcal{Y}| \times 1$
$(D+1) \times |\mathcal{Y}| \times 1$
Conv 128
ReLU
Conv 128
InstanceNorm
BatchNorm
ReLU
Conv 128
InstanceNorm
BatchNorm
ReLU
6x
Conv 1
sigmoid
Normalisation
sampling weights
$p(\mathbf{y}|\mathbf{s};\mathbf{w})$
$1 \times |\mathcal{Y}| \times 1$

Fig. 3.5: CONSAC neural network architecture. Figure from [40].

3.6 Limits of state-of-the-art methods

The methods described in this chapter work on a set of points and the geometric relationship between them. We can define the geometric information in many ways: as the ratio of models explaining all the points at the same time (Sec. 3.3), i.e., the larger the number of models shared by two points, the more likely they belong to the same geometric model; or as density (Sec. 3.4), i.e., denser regions are more likely to contain inliers to the same model. We could also avoid explicitly crafting the geometric information we are going to use, we can let the model learn it by itself (Sec. 3.5). Many statistical approaches can be proposed to guide the sampling, but we claim that all of these will only consider one side of the coin: geometry. When points come from images and we know it in advance, we can leverage this information and change the way we act. In particular, we can use the image in the robust estimation itself to get which regions are worth investigating and discard regions that are not relevant. This way, there is no need to sample at least once all the points, which can be very inefficient when the outlier ratio is high. The major limitation of the methods seen in this chapter is the fact they are not specialized for images, they do not use the semantics of the points and just treat them the same way.

In this work, we want to exploit the semantics of points to extract a set of discrete probabilities that can guide the sampling phase in robust estimation. So, we devote the next chapter to introduce the two main approaches to image processing.

Chapter 4

Image processing

As described in Sec. 1.1, robust estimation is the solution to many Computer Vision tasks where the sets of input points have been extracted from a picture $I \in \mathbb{R}^{H\times W\times 3}$, or set of pictures. The final objective of this work is to show that we can take advantage of the picture I in the robust estimation itself. To this end, we devote this chapter to present two approaches to image processing: the traditional approach where we use a hand-crafted feature extractor, like SIFT (Sec. 4.1.1), to extract locally-distinct points and the data-driven approach where we let a model, like a *Convolutional Neural Network* (Sec. 4.2.5), learn by itself the relevant features for a given task. In recent years, the data-driven approach has overcome and sometimes even replaced the traditional approaches [30], but this has not yet happened in the case of robust fitting. We aim to make the two approaches work together in a single method rather than just training a model end-to-end. Since we know in advance that points come from a picture I, we describe how to use I to improve the sampling. We devote this chapter to the presentation of the traditional hand-crafted approach in Sec. 4.1 and the data-driven approach in Sec. 4.2.

4.1 Hand-crafted approach

4.1.1 Scale Invariant Feature Transform

The *Scale Invariant Feature Transform* (SIFT) [16] is a method to extract features[1] from images. These features have very appealing properties that make the method still used today: they are invariant to image scale and rotation, and partially invariant to illumination changes and distortion.

[1]By features we mean distinctive aspects of something.

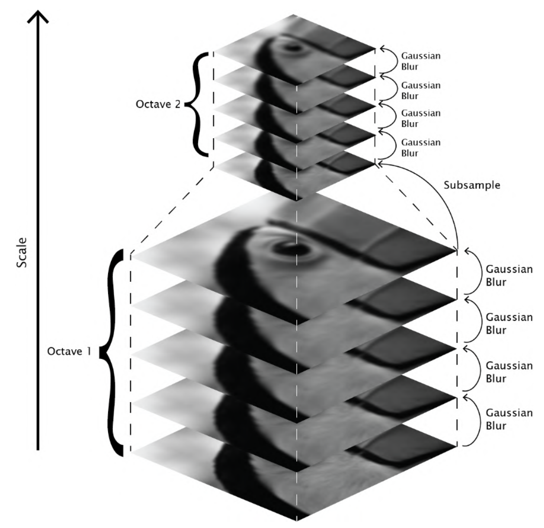


Fig. 4.1: Scale space pyramid. Figure from [43].

So, if we take two pictures of the same scene from two close locations, with different angles and lighting conditions, SIFT will produce similar features. SIFT features are encoded as a list of pairs of the form (*keypoint*, *descriptor*). A keypoint is a locally distinct point, i.e., a location of the image that differs from its neighborhood. A descriptor encodes the local appearance around the associated keypoint.

The SIFT algorithm consists of the following steps: first constructs the scale-space pyramid (Sec. 4.1.1.1) and computes the Difference of Gaussian (Sec. 4.1.1.2), secondly finds the candidate keypoints (Sec. 4.1.1.3) and filters out the ones representing edges (Sec. 4.1.1.4), thirdly finds the keypoints orientations (Sec. 4.1.1.5) and lastly generates the descriptors (Sec. 4.1.1.6).

4.1.1.1 Constructing the scale-space pyramid

To get rid of image details without adding new fake ones, SIFT takes the original image and generates five progressively blurred out images. Then, it resizes the original image to half size and generates blurred out images again. The process is repeated four times. Each set of five blurred images of the same size forms an *octave*. In practice, each blurred image L is obtained by convolving the image I with a Gaussian kernel G_σ having standard deviation σ:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{4.1}$$

where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.2}$$

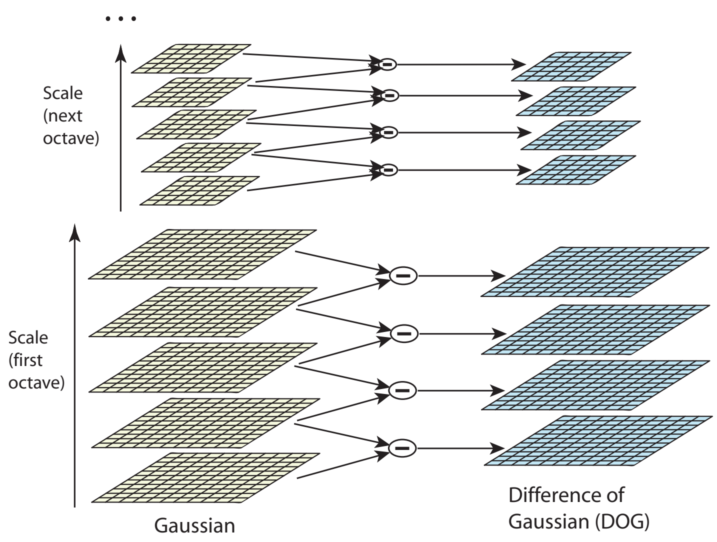


Fig. 4.2: Difference of Gaussian (DoG). Figure from [16].

As shown in Fig. 4.1, the series of blurred out and resized images, stacked on top of each other, forms the *scale-space pyramid.*

4.1.1.2 Compute the Difference of Gaussian

As shown in Fig. 4.2, for each octave, SIFT compute the difference $D(x, y, \sigma)$ between each pair of adjacent images separated by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{4.3}$$

The resulting images are called *Difference of Gaussian* (DoG) images and provide a close approximation to the *scale-normalized Laplacian of Gaussian*, which was shown to be required for true scale invariance [7].

As shown in Fig. 4.3, in DoG images only borders survive. The reason is that regions with the same intensity values remain the same when blurred out, while regions showing high diversity in intensity values undergo a significant transformation. These regions are the borders. In other words, pixels that have not changed due to blur will go to zero when subtracted, while pixels that changed a lot will survive.

4.1.1.3 Finding keypoints

As shown in Fig. 4.4, each pixel of $D(x, y, \sigma)$ is compared to its neighbors, both the eight pixels at the same scale and the nine in the scale above and

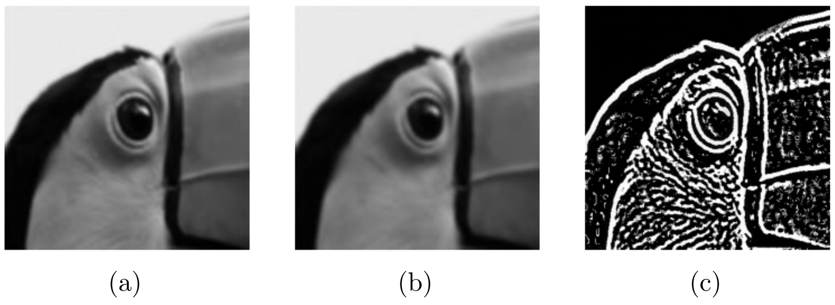


Fig. 4.3: (c) is the pixel-wise difference between (a) and (b). Notice that (a) and (b) have a different degree of blur. As you can see, borders bring up. Original figure from [43].

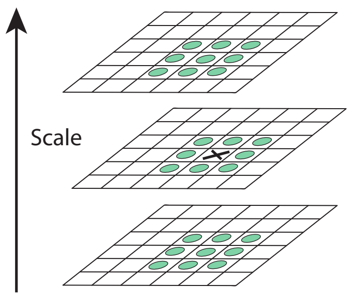


Fig. 4.4: Pixel-level extrema. The "X" marks the current pixel. The green bullets mark the neighbors. We mark "X" as candidate keypoint if it is larger or smaller than all his neighbors. Figure from [16].

below. The process is repeated for each DoG, except the topmost and the lowermost ones because they have not enough neighbors. In this way, we are able to detect local extrema[2] at pixel-level. Those are our candidate keypoints.

So far, extrema are pixels, i.e., the discretization of a continuous function which is the content represented by the image. So, instead of considering all points within the pixel as extrema, we can be more accurate by estimating the position of the extrema within the pixel. An example is shown in Fig. 4.5.

To compute sub-pixel extrema, we locally approximate $D(x, y, \sigma)$ in the position of each pixel-level extrema. This is done using the Taylor expansion up to the quadratic terms. Sub-pixel extrema increase the chances of matching and therefore the stability of the algorithm. In addition, to get rid of low-contrast keypoints, we remove those ones with magnitude $|D(\widehat{\mathbf{x}}, \widehat{\mathbf{y}}, \widehat{\sigma})| < 0.03$.

[2]By "extrema" we mean both maxima and minima.

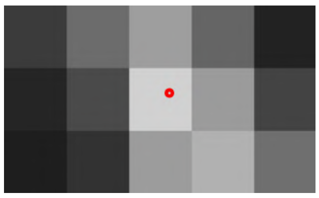
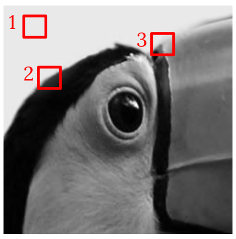
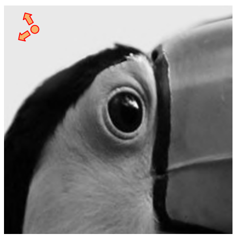


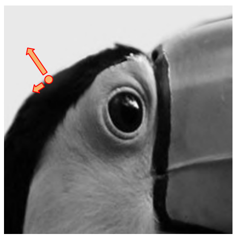
Fig. 4.5: The pixel-level maximum is the lighter pixel. The sub-pixel maximum is the red spot.



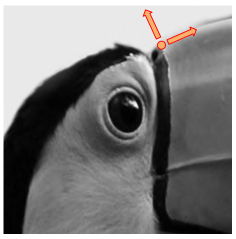
(a) Box 1 is a flat region; box 2 is an edge; box 3 is a corner.



(b) Principal curvatures of a flat region.



(c) Principal curvatures of an edge.



(d) Principal curvatures of a corner.

Fig. 4.6: Principal curvatures for flat regions, edges and corners. Original figure from [43].

4.1.1.4 Getting rid of edge keypoints

Fig. 4.6a shows that the image region around a keypoint can be flat, an edge or a corner. Let's analyze these cases:

- **Flat region.** The intensity values of the pixels are very similar, then both the principal curvatures[3] are small. It is shown in Fig. 4.6b.

- **Edges.** Across the edge the intensity values change, along the edge the intensity values are very similar. Therefore the principal curvature across the edge is large while the principal curvature along the edge is small. It is shown in Fig. 4.6c.

- **Corners.** The intensity values change in both directions, so both principal curvatures are large. It is shown in Fig. 4.6d.

Corners are locally distinguishable in both the directions, that is why we just want those keypoints while discarding edges. In practice, to eliminate edges,

[3]We do not provide a rigorous definition of *principal curvature*, we just remark that it measure how the surface bends in different directions at a point location.

SIFT uses a criterion based on the ratio of the eigenvalues of the Hessian:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4.4}$$

The derivatives are estimated by taking differences of neighboring pixels. The eigenvalues of **H** are proportional to the principal curvatures of D. We can avoid computing explicitly the eigenvalues as we are only interested in their ratio. Let α and β be eigenvalues of **H** with $\alpha > \beta$. Given a square matrix, the determinant is the product of the eigenvalues and the trace is the sum of the eigenvalues. So, we can compute the sum and the product of the eigenvalues directly from **H**:

$$\alpha + \beta = \mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} \tag{4.5}$$

$$\alpha\beta = \mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - D_{xy}^2 \tag{4.6}$$

Let's define the ratio $r = \alpha/\beta$ and use it to compute the following quantity:

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{4.7}$$

To sum up, we want

$$\underbrace{\frac{\alpha}{\beta}}_{r} < \text{threshold} \tag{4.8}$$

Since $(r+1)^2/r$ for $r > 1$ is a monotonic increasing function, we have

$$\frac{(r+1)^2}{r} < \frac{(\text{threshold}+1)^2}{\text{threshold}} \tag{4.9}$$

Using Eq. 4.7 we obtain

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(\text{threshold}+1)^2}{\text{threshold}} \tag{4.10}$$

SIFT uses a threshold equal to 10.

4.1.1.5 Finding keypoints orientation

To have rotation invariance, we need to assign an orientation to each keypoint. For each keypoint, we set a region size which depends on the scale: the bigger

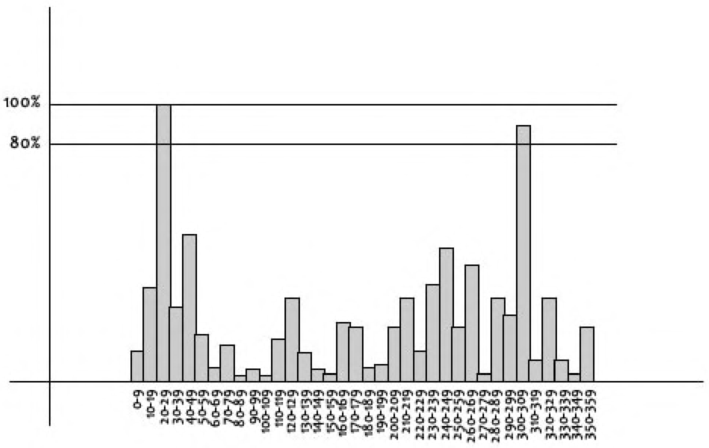


Fig. 4.7: Example of an orientation histogram. The histogram has peak at 20-29 degrees, so, we assign orientation 3 (the third bin) to the keypoint. Figure from [49].

the scale, the bigger the region. For all (x, y) within the region, gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are computed[4]:

$$m(x,y) = \sqrt{[L(x+1,y) - L(x-1,y)]^2 + [L(x,y+1) - L(x,y-1)]^2} \tag{4.11}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{4.12}$$

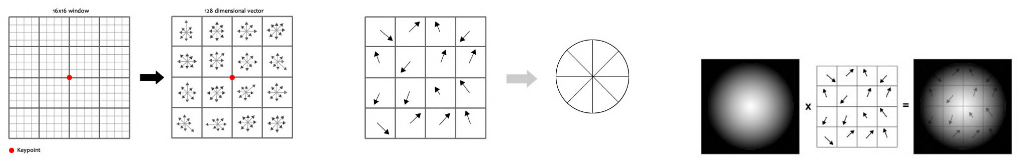
Once computed the gradient magnitude and orientation of the region, we build an orientation histogram with 36 bins covering 360 degree range of orientations. Before being added to the histogram, a gradient orientation is weighted by its gradient magnitude and by a Gaussian-weighted circular window with σ that is 1.5 times that of the scale of the keypoint.

Peaks in the histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected. Fig. 4.7 illustrates an example of the resulting orientation histogram.

Moreover, any other local peak above 80% of the highest peak is used to also create a new key point with that orientation (but same location and scale of the original key point). So, orientation can split up one key point into multiple key points.

Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.

[4]The scale of the key point is used to select the Gaussian smoothed image, L, with the closest scale, so that all computations are performed in a scale-invariant manner.



(a) Split 16×16 window into sixteen 4×4 windows

(b) Add gradient orientation into a 8 bin histogram.

(c) Multiply for the gaussian kernel.

Fig. 4.8: Visualization of some steps to compute the descriptors. Figure from [49].

4.1.1.6 Generating descriptors

First, we take a 16×16 window around the key point that we split into sixteen 4×4 windows. It is shown in Fig. 4.8a. For each 4×4 window we compute gradient magnitude and orientation. We add the gradient orientation to a histogram of 8 bins, each bin corresponding to 45 degrees. It is shown in Fig. 4.8b. Each gradient orientation is weighted by the corresponding gradient magnitude and by a Gaussian kernel that weight more the orientation closer to the keypoint. It is shown in Fig. 4.8c. Finally, we normalize the 128 values we have obtained from the 16×8 bins. To have rotation independence, the keypoint orientation is subtracted from each bin orientation. To have illumination independence, we threshold the values above 0.2. The resulting vector must be normalized again. This vector of length 128 is called *descriptor* and can uniquely identify a keypoint.

4.1.2 Feature Matching

Sec. 4.1.1 describes how to extracts the SIFT features from an input image. SIFT features are pairs representing locally distinct points in the descriptor space. Given an image pair, referred as left and right image, we independently extract the SIFT features, i.e., keypoints and descriptors. For each keypoint in the left picture, we get the most similar keypoint in right picture using *Nearest Neighbors* in the descriptor space.

As shown in Fig. 4.9, given a set of points X, Nearest Neighbor finds the closest point $\mathbf{x}_p \in X$ to a given query point $\mathbf{x}_q$. In the context of feature matching, we use Nearest Neighbor to get the set of corresponding points, or *matches*, $\left\{< \mathbf{x}_i^L, \mathbf{x}_i^R > \mid i = 1, \ldots, N\right\}$ where $\mathbf{x}_i^L$ and $\mathbf{x}_i^R$ represent the coordinates of the keypoints in the left and right picture, respectively[5].

To eliminate ambiguous matches we use the *ratio test* [16]. First of

[5]In this work, we denote with round brackets the point coordinates and with angle brackets the matches.

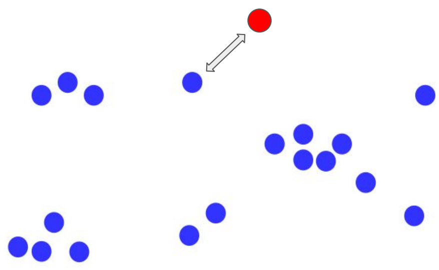


Fig. 4.9: Nearest Neighbor example. Given a set of blue points and a red query point, Nearest Neighbor selects the closest blue point.

all, instead of applying the Nearest Neighbor algorithm, we use a direct generalization called *k-Nearest Neighbor*, where k closest points are selected. For our purpose, we set $k = 2$ and we extract the two closest point in the right picture with respect to a given query point from the left picture. Then we discard the matches where the second-closest point is very close to the first. In particular, given a query point $\mathbf{x}_q$ and its two closest points in the descriptor space, denoted as $\mathbf{x}_i$ and $\mathbf{x}_j$, the ratio test tells us to discard the points that have the ratio of their distances to the query point greater than a threshold:

$$\frac{\|x_q - x_i\|_2}{\|x_q - x_j\|_2} > \varepsilon \tag{4.13}$$

where we set $\varepsilon = 0.8$ as suggested in [16]. Finally, to group points belonging to the same geometric transformation, we will use a robust estimator such as RANSAC (Sec. 2.2) or T-Linkage (Sec. 2.5). To ground our discussion on geometric transformations, Sec. 4.1.3 rigorously presents the elements we are going to need in this work.

4.1.3 Geometric transformations

In this section, we will describe the geometric transformation between points from two pictures. We start presenting the projective plane (Sec. 4.1.3.1). Then we will describe the homography transformation (Sec. 4.1.3.2) which maps a point in a picture to a point in the other picture. Then we will present a simple way to compute the homography given at least four corresponding points (Sec. 4.1.3.3). Then we will move to epipolar geometry (Sec. 4.1.3.5) so to introduce the fundamental matrix (Sec. 4.1.3.6) which maps a point in a picture to a line in the other picture. In short, the homography is used to map points of planar objects, while the fundamental matrix can relate points of 3D objects. Our objective is to find the transformation matrix, homography or fundamental matrix, given a set of corresponding points. For an exhaustive description, the reader is referred to [13].

4.1.3.1 The 2D projective plane

The projective plane $\mathbb{P}^2$ is a 3D space where two conditions are satisfied: the *homogeneous property*, i.e., $(a, b, c) = (ka, kb, kc)$, and $(0, 0, 0) \notin \mathbb{P}^2$. In $\mathbb{P}^2$ points and lines are both represented as 3D vectors. Let us see how to move them from $\mathbb{R}^2$ to $\mathbb{P}^2$. A point in $\mathbb{R}^2$ is represented by the pair (x, y) and for different values of x and y we have different points. We move to $\mathbb{P}^2$ by adding a third coordinate that we set to 1. Thus all the points of the form $(kx, ky, k)^\top$ for $k \neq 0$ are the same in $\mathbb{P}^2$ and we can always come back to $\mathbb{R}^2$ dividing by k and removing the 1 in the third coordinate[6]. A line in $\mathbb{R}^2$ is represented by the equation $ax + by + c = 0$. For different values of a, b and c we have different lines. We represent each line in $\mathbb{P}^2$ as the vector $(a, b, c)^\top$. As for points, all the lines of the form $(ka, kb, kc)^\top$ for $k \neq 0$ are the same line both in $\mathbb{P}^2$ (homogeneous property holds) and $\mathbb{R}^2$ (divide both sides of the equation by k). Having points and lines both represented as 3D vectors allows for very interesting results. Let us show three of them.

Result 4.1.1 *The point* $\mathbf{x}$ *lies on the line* $\mathbf{x}$ *if and only if* $\mathbf{x}^\top \mathbf{l} = 0$.

Result 4.1.2 *The intersection of two lines* $\mathbf{l}$ *and* $\mathbf{l}'$ *is the point* $\mathbf{x} = \mathbf{l} \times \mathbf{l}'$.

Result 4.1.3 *The line through two points* $\mathbf{x}$ *and* $\mathbf{x}'$ *is* $\mathbf{l} = \mathbf{x} \times \mathbf{x}'$.

4.1.3.2 Homography transformation

A *homography* (or *projective transformation*) is an invertible mapping $h : \mathbb{P}^2 \to \mathbb{P}^2$ such that three points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ lie on the same line if and only if $h(\mathbf{x}_1)$, $h(\mathbf{x}_2)$ and $h(\mathbf{x}_3)$ do. Moreover, for each homography h and point $\mathbf{x} \in \mathbb{P}^2$, there is a non-singular 3×3 matrix such that:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.14}$$

or more briefly, $x' = Hx$. We call H homogeneous matrix and it defines a mapping between a plane to another.

[6]When we add the third coordinate to a 2D point, we say it is in homogeneous coordinates.

4.1.3.3 Direct Linear Transformation

In this section, we are going to determine the homogeneous matrix H from a set of four corresponding points $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ where $\mathbf{x}_i = (x_i, y_i, z_i)^\top$ and $\mathbf{x}'_i = (x'_i, y'_i, z'_i)^\top$, $i \in [1,4]$. The projection of $\mathbf{x}_i$ using H is $\hat{\mathbf{x}}'_i$ and it is computed as:

$$\hat{\mathbf{x}}'_i = H\mathbf{x}_i = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} h^1\mathbf{x}_i \\ h^2\mathbf{x}_i \\ h^3\mathbf{x}_i \end{pmatrix} \tag{4.15}$$

where

$$h^k = \begin{pmatrix} h_{k1} & h_{k2} & h_{k3} \end{pmatrix} \tag{4.16}$$

Matrix H has to map $\mathbf{x}_i$ in $\mathbf{x}'_i$, thus $\mathbf{x}_i$ and its projection $\mathbf{x}'_i$ must be the same vector, i.e., $\lambda_1\mathbf{x}_i = \lambda_2\mathbf{x}'_i$ (recall that in projective geometry the homogeneous property hold, then $x = \lambda x \; \forall x$). This is equivalent to checking that $\mathbf{x}_i$ and $\mathbf{x}'_i$ are parallel vectors, which can be easily checked computing the cross-product of $\mathbf{x}_i$ and $\mathbf{x}'_i$ and imposing it equal to zero. So we obtain the following:

$$\mathbf{x}'_i \times \hat{\mathbf{x}}'_i = \begin{pmatrix} y'_i h^3\mathbf{x}_i - z'_i h^2\mathbf{x}_i \\ z'_i h^1\mathbf{x}_i - x'_i h^3\mathbf{x}_i \\ x'_i h^2\mathbf{x}_i - y'_i h^1\mathbf{x}_i \end{pmatrix} = \underbrace{\begin{pmatrix} 0^\intercal & z'_i\mathbf{x}_i^\intercal & -y'_i\mathbf{x}_i^\intercal \\ -z'_i\mathbf{x}_i^\intercal & 0^\intercal & -x'_i\mathbf{x}_i^\intercal \\ y'_i\mathbf{x}_i^\intercal & -x'_i\mathbf{x}_i^\intercal & 0^\intercal \end{pmatrix}}_{A_i\ (3\times 9)} \underbrace{\begin{pmatrix} h^{1\intercal} \\ h^{2\intercal} \\ h^{3\intercal} \end{pmatrix}}_{h\ (9\times 1)} \tag{4.17}$$

We have a system of linear equations in the unknowns $h_{11}, \cdots, h_{33}$. The third row is linearly independent and since the equations hold for any representation of $\mathbf{x}_i$, we can choose $z_i = 1$. So we can rewrite:

$$A_i = \underbrace{\begin{pmatrix} 0^\intercal & z'_i\mathbf{x}_i^\intercal & -y'_i\mathbf{x}_i^\intercal \\ -z'_i\mathbf{x}_i^\intercal & 0^\intercal & -x'_i\mathbf{x}_i^\intercal \end{pmatrix}}_{(2\times 9)} \tag{4.18}$$

Given a set of four corresponding point, we can build a system of 8 equations in 9 constraints:

$$\underbrace{\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix}}_{A\ (8\times 9)} \underbrace{\begin{pmatrix} h^{1\intercal} \\ h^{2\intercal} \\ h^{3\intercal} \end{pmatrix}}_{h\ (9\times 1)} = Ah \tag{4.19}$$

We can get an additional equation imposing $\|h\| = 1$. To obtain H, we compute the singular value decomposition of A:

$$A = \underbrace{U}_{(8\times 8)} \underbrace{D}_{(8\times 9)} \underbrace{V^\intercal}_{(9\times 9)} \tag{4.20}$$

The last row of $V^\intercal$ is $h = \begin{pmatrix} h_{11} & \cdots & h_{33} \end{pmatrix}$. To get H, we just need to reshape from 9×1 to 3×3.

4.1.3.4 Normalization phase

As a final remark on the DLT method, we are going to present the data normalization step. Data normalization is an essential pre-processing step in the DLT algorithm [13]. We can normalize the data points as follows:

1. Translate so that the centroid is at the origin (zero-mean points).
2. Scale so that the average distance from the origin is $\sqrt{2}$.
3. The transformation is independently applied to each of the two images.

In practice the transformation is carried out multiplying each point by a proper transformation matrix T. In particular T first translate the points and then scale them, thus it has the form:

$$T = \begin{pmatrix} s & 0 & s(-m_x) \\ 0 & s & s(-m_y) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.21}$$

where

$$m_x = \frac{1}{N}\sum_i^N x_i \qquad m_y = \frac{1}{N}\sum_i^N y_i$$

$$s = \frac{\sqrt{2}}{d_{\text{avg}}} \qquad d_{\text{avg}} = \frac{1}{N}\sqrt{(x_i - m_x)^2 + (y_i - m_y)^2}$$

Since we have two images, we have two vectors of points $\mathbf{x}$ and $\mathbf{x}'$, thus we are going to have two transformation matrices T_1 and T_2. We get the normalized points as:

$$\widetilde{\mathbf{x}}_i = T_1\mathbf{x}_i \qquad \widetilde{\mathbf{x}'}_i = T_2\mathbf{x}'_i \tag{4.22}$$

Applying DLT to $\widetilde{\mathbf{x}}_i$ and $\widetilde{\mathbf{x}'}_i$ we get $\widetilde{H}$. From $\widetilde{H}$ we get H as follows:

$$H = T_2^{-1}\widetilde{H}T_1 \tag{4.23}$$

4.1.3.5 Epipolar geometry

When we take a picture, the 3D scene is projected onto a 2D space and we lose the depth information. How can we get this information back? Human beings have two eyes to understand and grasp the depth of the world. By

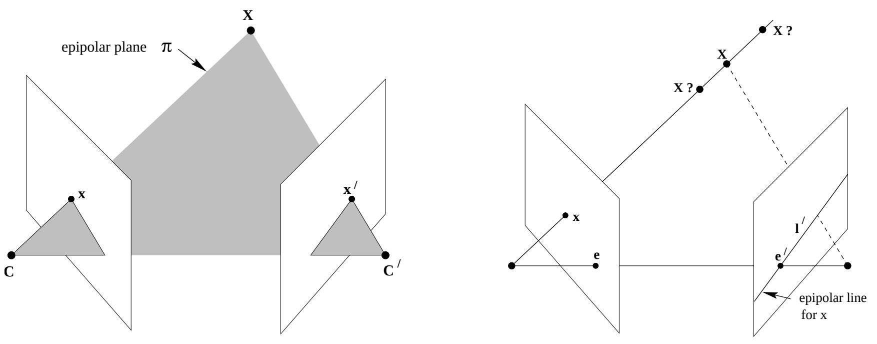


Fig. 4.10: (a) shows a basic setup with two cameras C and C', a 3D point $\mathbf{X}$ and its projections $\mathbf{x}$ and $\mathbf{x}'$ on the left and right pictures respectively. The term *projection* means the intersection between the left (right) picture and the ray defined by the 3D point $\mathbf{X}$ and the camera center C (C'). The plane π where C, C', $\mathbf{X}$, $\mathbf{x}$ and $\mathbf{x}'$ lie is called *epipolar plane.* (b) shows a 2D point $\mathbf{x}$ in the left picture that could be the projection of any point on the ray defined by $\mathbf{x}$ and the camera center C. By projecting each candidate 3D point $\mathbf{X}$ onto the right picture, we realize that all the possible candidate 2D points $\mathbf{x}'$ lie on the epipolar line $\mathbf{l}'$. This is why we say that each point $\mathbf{x}$ in the left picture is mapped into the corresponding epipolar line $\mathbf{l}'$ in the right picture: $\mathbf{x} \mapsto \mathbf{l}'$. The reasoning is symmetrical for the two pictures, so $\mathbf{x}' \mapsto l$ also holds. Figure from [13].

emulating humans, we can use two cameras, so we can extract 3D information by examining the relative positions of the objects in the two pictures, called *stereo images.* This practice is called *stereo vision. Epipolar geometry* is the geometry that relates the 3D points to their projections onto the 2D pictures. Fig. 4.10a shows a point $\mathbf{X}$ in the 3D space. There are two cameras C and C' taking a picture of the same scene from two different locations[7]. Thus the 3D point $\mathbf{X}$ is projected onto two 2D pictures at $\mathbf{x}$ and $\mathbf{x}'$.

To continue the presentation of epipolar geometry, let's introduce some terminology. The line joining C and C' is called *baseline.* The 3D point $\mathbf{X}$, the 2D points $\mathbf{x}$ and $\mathbf{x}'$, and the cameras C and C' lie on the same plane π, called *epipolar plane.* The intersections of π with the two pictures are two lines $\mathbf{l}$ and $\mathbf{l}'$ called *epipolar lines.* The baseline intersects the two pictures in two points $\mathbf{e}$ and $\mathbf{e}'$, called *epipoles.*

Suppose we only know a 2D point $\mathbf{x}$ in the left picture, while the corresponding 3D point $\mathbf{X}$ and its projection $\mathbf{x}'$ on the right picture are unknown. Fig. 4.10b shows that the 3D point $\mathbf{X}$ may lie everywhere on the ray defined by the camera center C and $\mathbf{x}$. By projecting each candidate 3D point $\mathbf{X}$ onto the right picture, we realize that all the possible candidate 2D points $\mathbf{x}'$

[7]C and C' are the two camera centers.

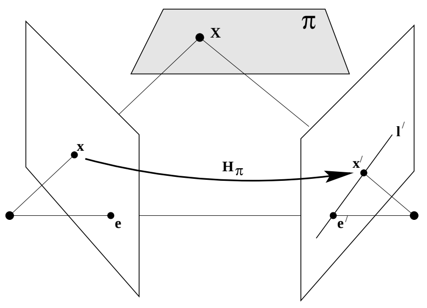


Fig. 4.11: π is a plane not passing through either the two camera centers. Point $\mathbf{x}$ in the left picture is projected onto π in a 3D point X that is then then projected to a point $\mathbf{x}'$ in the right picture. Plane π induce an homography H_π between the two pictures: each point in the left picture is mapped to a point in the right picture. Figure from [13].

lie on the epipolar line $\mathbf{l}'$. This is why we say that each point $\mathbf{x}$ in the left picture is mapped into the corresponding epipolar line $\mathbf{l}'$ in the right picture: $\mathbf{x} \mapsto \mathbf{l}'$. The reasoning is symmetrical for the two pictures, so also $\mathbf{x}' \mapsto l$ holds. The mapping from points to lines is represented by a matrix F, called *fundamental matrix*, presented in the next section (Sec. 4.1.3.6).

4.1.3.6 Fundamental matrix

As described in Sec. 4.1.3.5, given a pair of stereo images, each point in a picture is mapped into the corresponding epipolar line on the other picture and the fundamental matrix F describe this mapping. In the following, we geometrically derive the fundamental matrix.

Suppose again we only know a 2D point $\mathbf{x}$ in the left picture. Let's consider a plane π not passing through either the two camera centers. The intersection of π with the ray defined by $\mathbf{x}$ and the camera center C is a 3D point $\mathbf{X}$. This point $\mathbf{X}$ is then projected to a point $\mathbf{x}'$ in the right picture. Note, point $\mathbf{x}'$ must lie on the epipolar line $\mathbf{l}'$. Since the mapping from $\mathbf{x}$ to $\mathbf{x}'$ only depends on the intersection of planes and lines, plane π induce an homography H_π between the left and right pictures:

$$\mathbf{x}' = H_\pi \mathbf{x} \tag{4.24}$$

Since both the epipole $\mathbf{e}'$ and the point $\mathbf{x}'$ must lie on the epipolar line $\mathbf{l}'$, by using Result 4.1.3 we compute $\mathbf{l}'$ as:

$$\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{x}' \tag{4.25}$$

where

$$[\mathbf{e}']_\times = \begin{pmatrix} 0 & -\mathbf{e}'_z & \mathbf{e}'_y \\ \mathbf{e}'_z & 0 & -\mathbf{e}'_x \\ -\mathbf{e}'_y & \mathbf{e}'_x & 0 \end{pmatrix} \tag{4.26}$$

with $\mathbf{e}' = (\mathbf{e}'_x, \mathbf{e}'_y, \mathbf{e}'_z)^\top$. Combining Eq. 4.24 and Eq. 4.25 we get:

$$\mathbf{l}' = [\mathbf{e}']_\times \mathbf{x}' = \underbrace{[\mathbf{e}']_\times H_\pi}_{F} \mathbf{x} = F\mathbf{x} \tag{4.27}$$

where F is the fundamental matrix. Note, plane π is used as a means of defining a point map from one picture to another, but it is not required in order for F to exists.

In the following, we are going to prove an important property for the fundamental matrix. Given a pair of corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$, since $\mathbf{x}'$ must lie on the epipolar line $\mathbf{l}'$, by using Result 4.1.1, we get:

$$\mathbf{x}'^\top \mathbf{l}' = 0 \tag{4.28}$$

From Eq. 4.28 and Eq. 4.27 we get the following system of equations:

$$\begin{cases} \mathbf{l}' = F\mathbf{x} \\ \mathbf{x}'^\top \mathbf{l}' = 0 \end{cases} \tag{4.29}$$

By substituting the first equation into the second we get Result 4.1.4.

Result 4.1.4 *Given a pair of corresponding points* $\mathbf{x} \leftrightarrow \mathbf{x}'$ *in the two pictures, the fundamental matrix satisfies the following condition:* $\mathbf{x}'^\top F\mathbf{x} = 0$

It gives a way of characterizing the fundamental matrix only in terms of corresponding points in the pictures. This enables F to be computed from picture correspondences alone. We can compute the fundamental matrix F in a similar way to what we did with homography by solving a system of equations. To uniquely determine the parameters of F we need at least eight points.

4.2 Data-driven approach

4.2.1 Introduction to Machine Learning

Machine Learning (ML) is a sub-field of *Artificial Intelligence* concerned with the question of how to build *algorithms* and *models* that automatically improve with experience. By model we mean a parametric function that takes an input sample and returns an output. Based on the type of samples and the desired output, we distinguish different *learning paradigms*, such as *Supervised Learning*, *Unsupervised Learning* and *Reinforcement Learning.* We say a model is *learning* if it improves its performance (with respect to a given metric and task) while looking at the data. The goal of ML is to learn the model[8] directly from the data.

4.2.2 Supervised Learning

In *Supervised Learning* (SL) the model learns the mapping between the input $\mathbf{x}_i$ and its desired output (or *target*) t_i. We can identify three phases: *training*, *testing* and *production.* During the training phase, the model looks through a data-set, called *training-set*, and learns, meaning tune its parameters. During the testing phase, we assess the model measuring its performances on a new data-set, called *test-set.* The reason why we need a new data-set is that we are not interested in the model's ability to store information, but its ability to correctly predict the output of new samples that it has never seen before. This capability is called *generalization.* During the production phase, the model already tested is used to predict the output of a given input.

The model is trained so that the performances are maximized, this corresponds to minimizing an error (or loss) function that is a measure of the distance between the desired output and the output provided by the model. The training-set consists of pairs: $TR = \{(\mathbf{x}_i, t_i) \mid i \in [1, N]\}$ and based on the domain of the target t_i, we can define two problems:

- **Regression.** The target is a continuous value, i.e., $t_i \in \mathbb{R}$.
- **Classification.** The target belongs to a finite set of discrete categories (also called *labels* or *classes*), i.e., $t_i \in \Lambda = \{\lambda_1, ..., \lambda_N\}$.

Artificial Neural Networks are a successful class of non-linear models used in SL and they are going to be described in the next section (Sec. 4.2.3).

4.2.3 Artificial Neural Networks

An *Artificial Neural Network* (ANN) is weighted directed graph whose vertices are called *neurons* and whose edges are called *connections.* Neurons that receive the input from the environment are called *input neurons*, while neurons that emit the result to the environment are called *output neurons.* The remaining neurons that have no contact with the environment (but only with

[8]Learning a model means tuning its parameters.

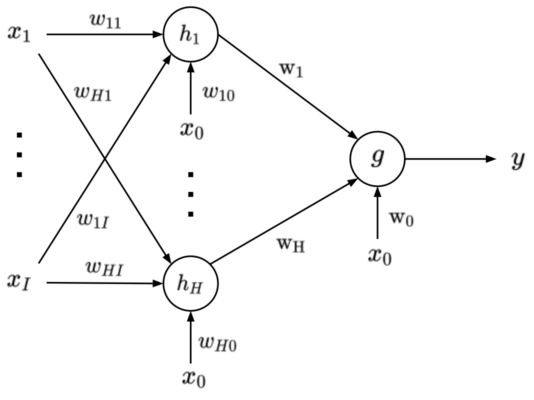


Fig. 4.12: Feed Forward Neural Network with I input neurons, 1 output neuron and a single hidden layer with H neurons. h_j and g are nonlinear function, called *activation functions.*

other neurons) are called *hidden neurons.* Neurons are grouped into layers, forming an *input layer*, an *output layer* and several *hidden layers* [34]. ANN are universal approximating functions, i.e., they are able to approximate any continuous function over a compact domain at any required level of precision (by increasing the number of its parameters) [2] [4] [6] [10] [37] [42]. A special case of ANN is the *Feed Forward Neural Network* (FFNN) where the graph does not contain loops.

Fig. 4.12 shows a simple FFNN implementing the following function:

$$y = g\left(\sum_{j=0}^{H} W_j \; h_j\left(\sum_{i=0}^{I} w_{ji}x_i\right)\right) \tag{4.30}$$

where $x_0 = 1$, $h_0(\cdot) = 1$, h_j and g are nonlinear function, called *activation functions.* The network takes in input a I-dimensional point $\mathbf{x} = (x_1, ..., x_I)$ and then each neuron computes a nonlinear function of the linear combination of its inputs. The results are propagated from left to right enabling the computation for the consecutive layer.

The activation function used by an hidden neuron can be a *sigmoid*, an *hyperbolic tangent* or a *Rectified Linear Unit*, shown respectively in Fig. 4.13a, Fig. 4.13b and Fig. 4.13c. The activation function used by the output neuron depends on the task. In regression we can simply use a linear function. In classification we can use the sigmoid or, when we need to interpret the output of the network as a probability, we can use the *softmax* activation function:

$$\mathrm{softmax}_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{4.31}$$

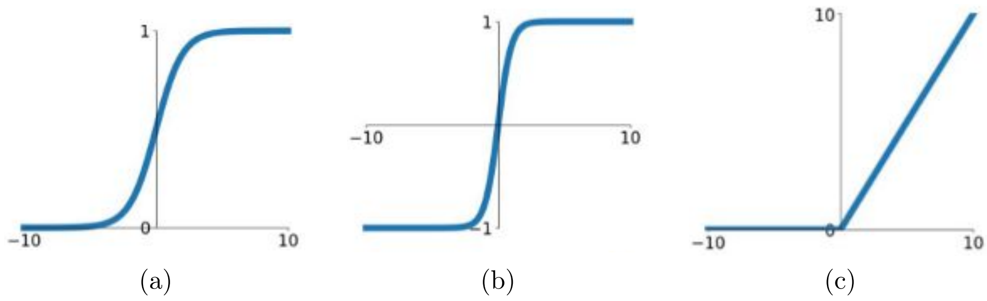


Fig. 4.13: (a) is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$; (b) is the hyperbolic tangent function $\tanh(x)$; (c) is the Rectified Linear Unit function $\text{ReLU}(x) = \max(0, x)$.

where $i = 1, \ldots, C$ and C is the number of classes. We use the softmax to normalize the outputs values so that they sum to 1.

A FFNN is trained using *error backpropagation*, i.e., the error is iteratively minimized using the following update rule:

$$w^{k+1} \leftarrow w^k - \eta \left[\frac{\partial E}{\partial w}\right]_{w=w^k} \tag{4.32}$$

where E is the error (or loss) function. Back-propagation consists of two steps:

- **Forward pass.** We compute the output of every neuron, from the input to the output layer.
- **Backward pass.** We start from the output layer and going backward to the input layer we recursively apply the *chain rule* to update the weights.

In regression, the error is usually the sum of the squared error (Eq. 4.33), while in classification it is usually the cross entropy loss (Eq. 4.34).

$$E = \sum_{n=1}^{N} (t_n - y_n)^2 \tag{4.33}$$

$$E = \sum_{n=1}^{N} (t_n \log(y_n) + (1 - t_n)\log(y_n))^2 \tag{4.34}$$

A FFNN trained for classification is called *classifier*.

4.2.4 Image classification and localization

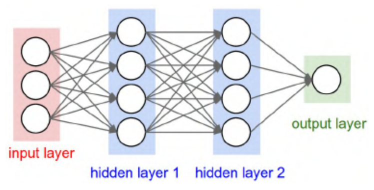
Given an input image $I \in \mathbb{R}^{H \times W \times 3}$ and a fixed set of *labels* (or *classes*) $\Lambda = \{\lambda_1, ..., \lambda_N\}$, *image classification* is the problem of assigning a label λ_i to image I. It is a specific case of the classification problem described in Sec. 4.2.1 where the objects of the classification are images. We can further divide between *multi-class classification* where we assume each image is assigned to a single target label and *multi-label classification* where the image contains many objects of different classes and can be assigned to more than one target label. The *localization* problem, in addition to the label, returns the coordinates (x, y, h, w) of the bounding box enclosing that object. In order to make the classification, we feed the classifier with a number of relevant features that represent some properties of the image, such as a distinct shape (edge, corner, curve, or even an object) or a color. Note, a good feature selection improves the classification accuracy and generalization. We can broadly identify two families of features: *hand-crafted* and *learned* features. Hand-crafted features are manually designed, engineered, and selected by experts in the field. Representatives of this family are the SIFT features, described in Sec. 4.1.1. In some cases, experts may not be able to identify which features are optimal in a given task and may select sub-optimal features. Learned features solve this issue. The idea is to let the model learn on its own what the relevant features are by looking at many examples. Feature selection and classification are jointly carried out by a single model, such as a *Convolutional Neural Network*, described in Sec. 4.2.5. When the network learns both the features and the classifier out of it, we talk about *Deep Learning* (DL) [30].

There are pros and cons for both families. Hand-crafted features have good properties that learned features lack: they allow us to leverage prior information, are interpretable, and don't require a large data-set to be designed. On the other hand, having a model capable of selecting by itself the optimal features for a given task leads to better results.

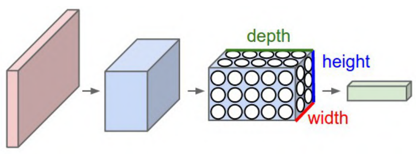
In the next section (Sec. 4.2.5) we are going to describe Convolutional Neural Networks.

4.2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [3] are a special type of ANNs that make the explicit assumption that inputs are images. CNNs consists of two blocks: a *feature extractor* (FE) and a *classifier*. The FE mainly uses two types of layers: convolutional and max-pooling layers. Convolutional layers, unlike fully-connected layers, have neurons arranged in three dimensions:



(a)



(b)

Fig. 4.14: (a) is a regular 3-layer ANN; (b) is a CNN whose neurons are arranged in three dimensions width, height and depth. The red input layer holds the image, so its width and height would be the dimension of the image and depth would be 3 (RGB image). Figures (a) and (b) from [53].

width (w), height (h) and depth (d), as shown in Fig. 4.14. We call *depth slice* the set of $w \times h$ neurons with the same depth $\bar{d}$.

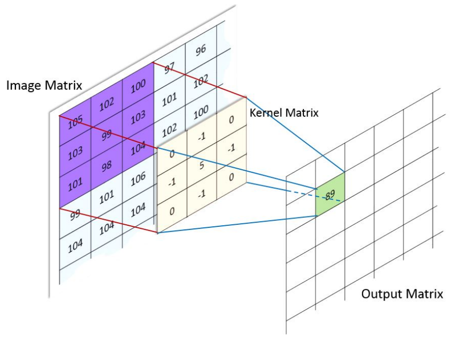
The assumption that inputs are images is encoded into the network using convolutional layers since they have two properties: the *local connectivity* and the *parameter sharing*. Local connectivity means that each neuron is connected to only a local region of the input volume[9]. Parameter sharing means that all the neuron in a depth slice are forced to use the same weights and bias. Every convolutional layer transforms a 3D input volume to a 3D output volume, usually by decreasing its spatial size (width and height) and increasing its depth. The convolutional layer's parameters consist of a set of learnable filters (or kernels). Every filter is small spatially but extends through the full depth of the input volume. During the forward pass, we slide each filter across the width and the height of the input volume and compute the dot product between the entries of the filter and the input at any position. This operation is called *convolution* and an example is shown in Fig. 4.15.

As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map (or feature map) that gives the responses of that filter at every spatial location. Each filter produce a separate 2-dimensional activation map. Intuitively, the network will learn filters that activate when they see some type of visual feature or pattern. We will stack this activation maps along the depth dimension and produce the output volume.

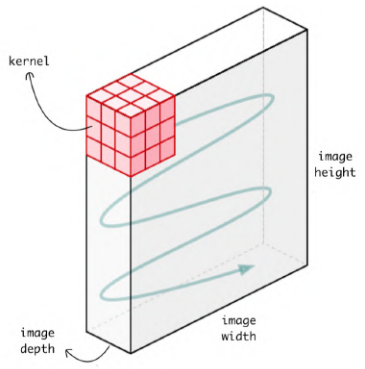
We call *receptive field* of a pixel[10] (p_x, p_y), the region of the input volume involved in the computation of (p_x, p_y). In other words, the receptive field is the region of the input that affects a pixel in the output map. Usually, the term receptive field refers to the regions of the network input that affect the

[9] Recall that in fully-connected layers each neuron is connected to the entire input.

[10] As for the locations in the image, we refer to a location in the activation map with the term *pixel*.



(a)



(b)

Fig. 4.15: In (a), we compute the element-wise product between the filter (or kernel) and the overlapped subset. The result is summed up to get a single value. In (b), both the filter and the overlapping region are shown to be 3D volumes, rather than 2D arrays. In addition, the blue arrows show the filter sliding. Figure (a) from [46]. Figure (b) from [52].

score of an output neuron.

The depth of the output volume corresponds to the number of filters, while the spatial dimension may be computed using the formula:

$$1 + \frac{W - F + 2P}{S} \tag{4.35}$$

where W and F are respectively the input and the filter width, S is the stride (number of pixels we move the filter while sliding) and P is the size of the zero-padding (to control the input dimension, we surround it with zeros).

As previously mentioned, in addition to convolutional layers, we also have max-pooling layers. They are periodically put in-between successive convolutional layers to reduce the spatial size of the input. They operates independently on every activation map resizing it spatially using the *max* operation. An example is shown in Fig. 4.16.

The max pooling layer works as a switch in the backward pass routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass the pooling layer keep track of the index of the max activation. Moreover, the pooling operation provides a certain degree of distortion invariance because reducing the size, images with shifted pixels results similar in the reduced representation.

Finally, at the top of the network there is the classifier which consists in one or more fully connected layers. Classifiers were described in Sec. 4.2.3.

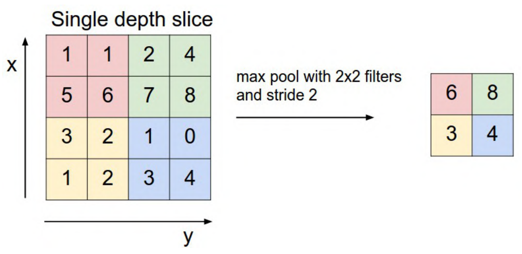


Fig. 4.16: Max pooling layer with filter size 2×2 applied with a stride of 2. It computes the maximum for each colored square region. The 75% of the activation pixels are discarded. Figure from [53].

4.2.6 Class Activation Mapping technique

As previously mentioned, learned features make the classification a black-box task, meaning that designers do not know what the CNN is looking at when performing the prediction. From the outside we can only see the mapping between inputs and outputs without understanding the reasons that drive the model to make a specific prediction.

Zhou *et al* [32] proposed a technique, called *class activation mapping*, that allows us to know not only the prediction of the target label, but also the corresponding relevant image regions that the CNN focuses on when it is predicting the output. As shown in the lower right of Fig. 4.17, a colored mask, called *class activation map* (CAM), is overlapped on the image to highlight the most important regions involved during the prediction. Red regions are very relevant, while blue ones are irrelevant. The shift from red to blue is gradual. That is why there are yellow (meaning relevant) and cyan (meaning slightly relevant) regions. This technique is mainly used for *explainability*, i.e., to visualize what is driving a given prediction but, since relevant regions of the image contain the target objects, we can use the technique also for *localization*. Convolutional layers are meaningful pattern detectors, but this capability is lost when fully-connected layers are used for classification. In order to preserve the localization ability, we can perform *global average pooling* (GAP) on the feature maps just before the classifier and uses those as features for the prediction. The network architecture is illustrated in the upper part of Fig. 4.17.

The score of a class λ, denoted by S_λ, with $\lambda \in \Lambda$, is computed as follows:

$$S_\lambda = \sum_k \left({w_k}^\lambda \frac{1}{W \cdot H} \sum_{x,y} f_k(x,y) \right) \tag{4.36}$$

where f_k with $k \in \{1, \ldots, K\}$ denotes the k-th feature map of the last convolutional layer; ${w_k}^\lambda$ is the class λ weight that multiplies f_k; $x \in \{1, \ldots, W\}$;

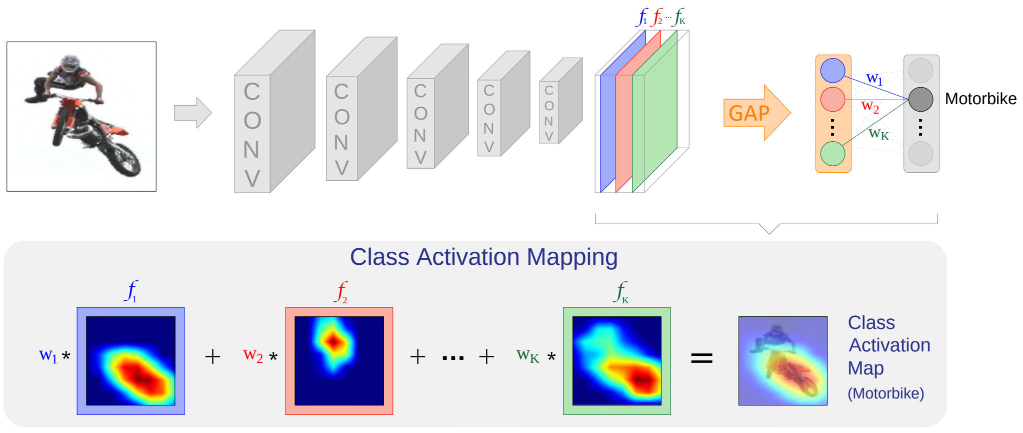


Fig. 4.17: In the upper part of the image the CNN architecture is shown. In the lower part the Class Activation Mapping technique is shown. Original figure from [32].

$y \in \{1, \dots, H\}$; W and H denote the width and the height of the features maps f_k. The class activation mapping technique is trained in a *Weakly Supervised Learning* (WSL) setting[11], meaning that only image-level labels are required as targets in the training-set. At inference time, we use the *class activation mapping* technique to produce the CAM[12] of the target class λ. The procedure is illustrated in the lower part of Fig. 4.17. The CAM of class λ, denoted by M_λ is computed as the weighted sum of the feature maps:

$$M_\lambda(x, y) = \sum_k {w_k}^\lambda f_k(x, y) \tag{4.37}$$

Moreover, we can show that, reorganizing Eq. 4.36, the score of class λ is equal to the spatial average of the CAM (of class λ):

$$\begin{aligned} S_\lambda &= \sum_k \left({w_k}^\lambda \frac{1}{W \cdot H} \sum_{x,y} f_k(x, y) \right) \\ &= \frac{1}{W \cdot H} \sum_{x,y} \underbrace{\left(\sum_k {w_k}^\lambda f_k(x, y) \right)}_{=: M_\lambda(x,y)} \\ &= \frac{1}{W \cdot H} \sum_{x,y} M_\lambda(x, y) \end{aligned} \tag{4.38}$$

[11]By WSL we mean training without ground-truth bounding boxes.

[12]The class activation map (CAM) is the output of the class activation mapping (CAM) technique. Procedure and output object are shortened in the same way.

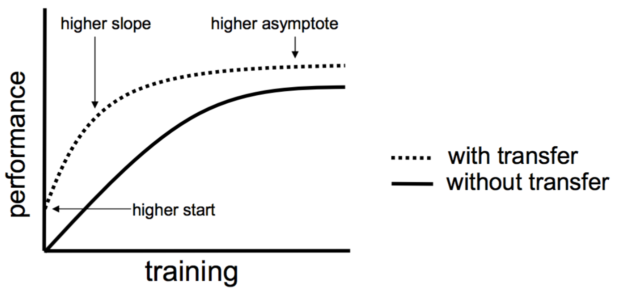


Fig. 4.18: Three ways Transfer Learning can improve performance during training. Figure from [22].

As a final remark, it is important to observe that the CAM method does not guarantee to localize the full extent of the objects: it highlights the regions it deems relevant but the map could cover sub-parts of the object or the even part of the background (coarse edges of the red regions) and that is why many methods, like [38] [36], were proposed to improve the quality of the CAMs. The reader may wonder why not directly train a network for object detection. In some tasks training in a *fully supervised* setting (with the bounding boxes around each object for every image) is not feasible, due to time constraints or monetary costs, that is why the weakly-supervised approach (use only image-level labels) is sometimes preferred. Furthermore, we are fascinated by the ability of the network to discover the intrinsic relationship between a label and a shape or pattern in the image. We train the network for image classification, so only labels are required.

4.2.7 Transfer Learning

Transfer learning is a method to reuse the feature extractor learned for a source task S in a destination task D. The idea is to first train a base CNN for task S using a training set TR^S, then remove the classifier and add a new classifier specific for the task D we want to tackle that we train using a new training set TR^D. The rationale behind transfer learning for image classification is that if a model is trained on a large and general enough data-set, the feature extractor will still able to extract good features representing elements of the visual world. We can exploit these learned feature maps for a new task, without having to start from scratch by training a large model on a large data-set. As described in [22] and shown in Fig. 4.18, there are three possible benefits when using transfer learning:

- **Higher start.** The initial performance is higher than it otherwise would be.

- **Higher slope.** The rate of improvement during training is steeper than it otherwise would be.
- **Higher asymptote.** The converged performance of the trained model is better than it otherwise would be.

If the destination data-set TR^D is large enough, it is still common practice to start from a pre-trained network and perform *fine-tuning*. The idea is to unfreeze a few of the top layers of a frozen base model and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to fine-tune the higher-level features in the base model in order to make them more relevant for the destination task [50].

Transfer learning is powerful but can yield poor results. The reason is that the features have been optimized for another problem (the source problem) and may be sub-optimal for the destination problem.

Chapter 5

Proposed Method

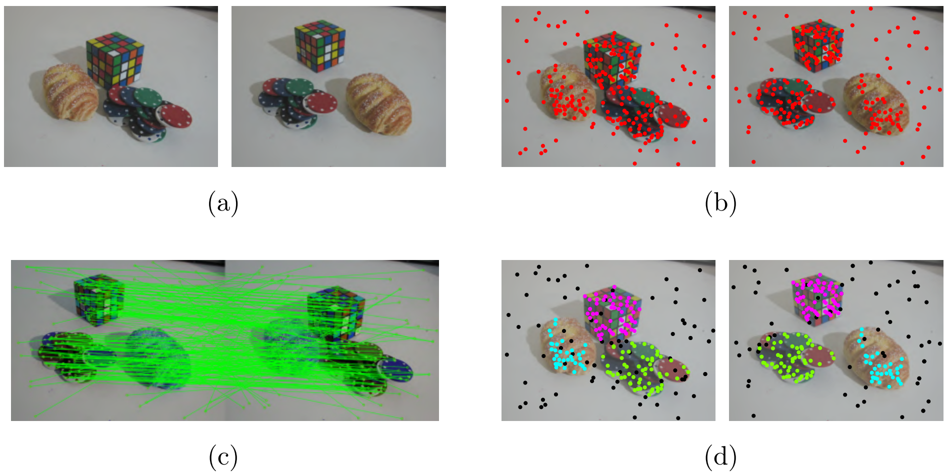

(a) (b) (c) (d)

Fig. 5.1: (a) shows an image pair depicting three objects: bread, cube and chips; (b) shows the keypoints marked in red; (c) show the point-matches between the pictures. (d) shows the clusters of points, each one represented with a different color. Original figure from AdelaideRMF data-set [25].

Deep learning (DL) has dramatically improved the state-of-the-art in image classification, localization and many other tasks [30]. Unfortunately, for tasks like robust multi-model fitting where little data is available, deep learning alone does not outperform the performance of classical robust estimators. In this work, we propose a DL-based sampling strategy to boost up robust estimators dealing with multi-view geometry problems. As described in Sec. 4.1.3.5, multi-view geometry is the study of the geometric relations between multiple views of a 3D scene. Typically, given two (or more) pictures of the same scene, as in Fig. 5.1a, we want to automatically estimate the geometric relationships between corresponding points in the images. We recall the process to get corresponding points from an image pair. We independently extract a set of points from each picture as in Fig. 5.1b. To this end, we can use a hand-crafted feature extractor like SIFT (Sec. 4.1.1). The points are matched across pictures (Fig. 5.1c) using Nearest Neighbors and ratio test (Sec. 4.1.2). We now have a set of corresponding points described by a series of geometric models and we can apply a robust estimator like T-Linkage (Sec. 2.5) to cluster points belonging to the same model and filtering outliers. Fig. 5.1a shows a dynamic scene where three objects (bread, cube and chips) move in different positions between the left and right image, thus there are three fundamental matrices that describe the scene, each corresponding to an object. The result of the robust estimation is shown in Fig. 5.1d. Note that traditional robust fitting approaches use underlying pictures as in Fig. 5.1a only to extract point correspondences and then discard it, the images are not used in the robust estimation itself.

Human observers can easily recognize the different objects in pictures. For example in Fig. 5.1a we can clearly distinguish the cube from other objects and background and by comparing it with Fig. 5.1b we can infer that all the red points that lie on the cube image-region have undergone the same rigid motion, i.e., they are inliers of the fundamental matrix describing the motion of the cube. The presence of semantics helps humans to group point correspondences and the main intuition behind the proposed approach is that a semantic prior can be helpful also in robust estimation. To this end, to extract the semantic information from the image, we use a Convolutional Neural Network (CNN) which has been shown to be an optimal pattern locator [32]. The CNN maps the visual appearance of an object to a label which represents its meaning and that is why we refer to this information with the term *semantics*. We will use the semantics to guide the sampling phase so to discover geometric models.

The remaining of this chapter is organized as follows. Sec. 5.1 describes how to extract semantics from an image couple to produce a set of discrete probabilities, called *semantic priors*, that robust estimators use to guide the sampling phase and discover geometric models in the data. This novel sampling strategy, called *semantic-aware sampling*, is described in Sec. 5.2. Moreover in Sec. 5.3 we describe a thresholding technique to filter outliers.

5.1 Semantic prior

As mentioned in the introduction to this chapter, we use information extracted from the Convolutional Neural Network (CNN) to guide the sampling towards

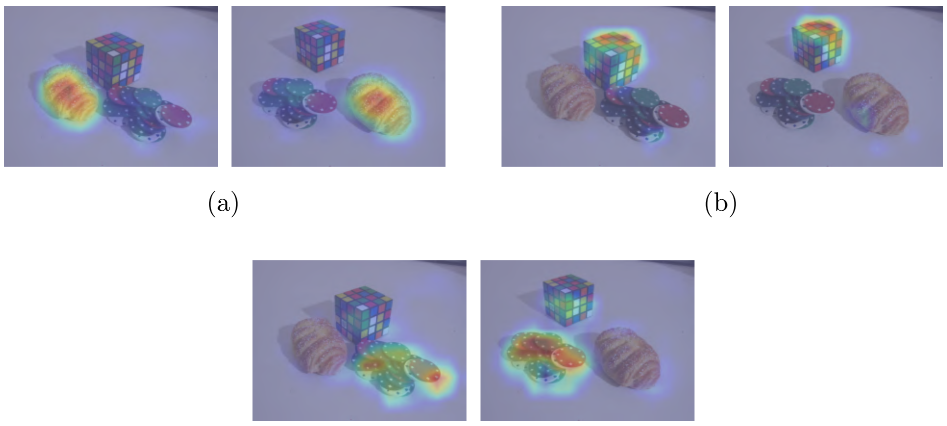


Fig. 5.2: Each CAM locates the patterns of the corresponding object: (a) locates the bread; (b) locates the cube and (c) locates the chips. Original figure from AdelaideRMF data-set [25].

good geometric models. Specifically, we propose two approaches: the first leverages the Class Activation Mapping (CAM) method while the second exploits the features coming from the last convolutional layer of the CNN.

We have selected these two approaches because both allow us to get the location of the objects (or significant patterns) in the image without being trained using ground-truth bounding boxes. We train a CNN for image classification having in mind the idea that in order to map images to labels, the network intrinsically learns a high-level representation of the image objects and we can exploit it for localization. The CAM method returns an aggregate map that highlights the patterns of a single object but requires that the classes the image can belong to are fixed. When we expose the model with an image of an unknown class, the classifier cannot guess the target label (simply because it is unknown) but, as long as the patterns of the depicted object are similar to one or more patterns known by the model, the feature maps can localize the patterns in the image. In practice, sometimes we have good feature maps but aggregating them using the weights of the classifier produce poor CAMs unable to localize the object in the image. This happens because the object does not belong to any of the fixed classes, thus we decided to directly use the features from the last convolutional layer (before being aggregated). On the one hand we no longer know which pattern each map represents, on the other hand we are extending the applicability of the method to patterns similar to those learned from the network but mapped to unknown labels.

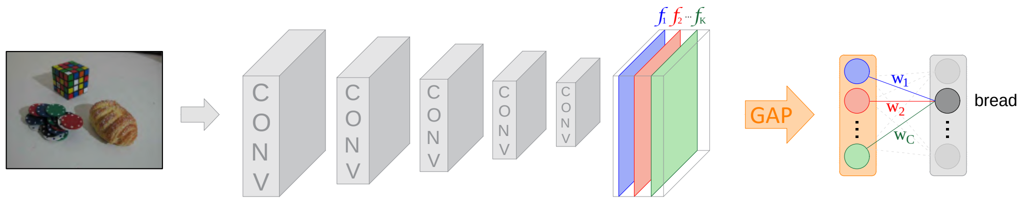


Fig. 5.3: CNN architectures to extract semantics by using the CAM method. The output of the last convolutional layer is a tensor composed by K feature maps $\{f_1, \dots, f_K\}$. We use a Global Average Pooling (GAP) Layer to spatially aggregate the pixels of each f_k and we feed the resulting vector to the classifier. Original figures from [32] and AdelaideRMF data-set [25].

5.1.1 Semantic prior from CAMs

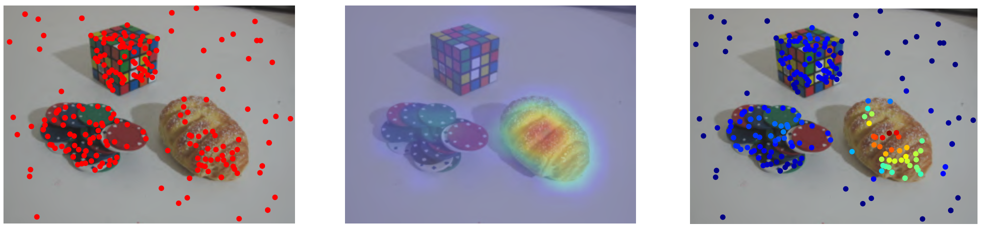
Our first approach uses the CAM method (Sec. 4.2.6) in a multi-label classification setting, i.e., the model associates the input image $I \in \mathbb{R}^{H \times W \times 3}$ with a set of labels $\bar{\Lambda} \subseteq \Lambda$ corresponding to the objects in the image rather than a single label. The set $\bar{\Lambda}$ will contain all the labels that have probability greater than a threshold, e.g., 0.5. For example, from Fig. 5.1a we get $\bar{\Lambda} = \{\text{bread}, \text{cube}, \text{chips}\}$. For each label $\lambda \in \bar{\Lambda}$, we produce a matrix $M_\lambda \in \mathbb{R}^{H \times W}$ whose values are larger in the regions where the patterns of object λ are located. M_λ is called Class Activation Map (CAM) and localizes objects of class λ in the picture [32]. An example is shown in Fig. 5.2. We can interpret each pixel of the CAM M_λ as the probability p that the corresponding pixel in the image contains an object of class λ. The idea is to assign this probability to the points we are going to sample thus making the sampling of pure minimal sample sets (MSS) more likely. Given a set of points $X = \{(x_n, y_n) \mid n = 1, \dots, N\}$, we define the n-th entry of the discrete probability p as:

$$p_n = M_\lambda(x_n, y_n) \tag{5.1}$$

where $n \in \{1, \dots, N\}$. Note that, before being used, p needs to be normalized so that the probabilities sum to 1:

$$\sum_{n=1}^{N} p_n = 1 \tag{5.2}$$

An example of this process is shown in Fig. 5.4 where we embed the semantics of the bread into the points so to obtain a discrete probability vector p that we use to guide the estimator to sample points belonging to the bread. Note, the example shows that CNNs are excellent pattern locators [32] and can localize the bread pattern very well in the picture.



(a) (b) (c)

Fig. 5.4: Embed semantics into discrete probability. (a) shows the points over the picture; (b) show the CAM M_λ with $\lambda =$ “bread”; (c) graphically represents the discrete probability vector p. Original figure from AdelaideRMF data-set [25].

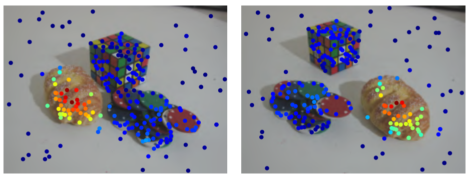
We further extend this basic process as we have a couple of pictures representing the same scene (Fig. 5.1a) and we are interested in sampling good matches rather than points per se. Therefore we impose an additional constraint: the probability of a match is the minimum between the probabilities of the match end-points. We denote as p^L and p^R the left and right discrete probability vectors corresponding respectively to the left and right image. We compute the n-th entry of the discrete probability vector as:

$$\psi_n = \min\{p^L_n, p^R_n\} \tag{5.3}$$

Note that, also in this case, before being used, ψ must be normalized so that the probabilities sum to 1.

$$\sum_{n=1}^{N} \psi_n = 1 \tag{5.4}$$

where $n \in \{1, \dots, N\}$. A comparison from the disjoint probabilities (p^L and p^R) and the joint probability ψ is graphically represented in Fig. 5.5: all the points that lie over the bread have high disjoint probabilities p^L and p^R (Fig. 5.5a) while some of this points have low joint probability ψ (Fig. 5.5b). This happens to wrong correspondences that match point on the bread to background points in the other image. With the mechanism described so far, we jointly use the semantics of both the left and right picture and this allows to naturally discard mismatches by assigning them low probability of being sampled. We can now repeat the process for the CAMs of the other labels in $\bar{\Lambda}$ and we obtain three discrete probabilities that can be used as priors to guide the samplings. The semantic priors of the cube and chips are shown in Fig. 5.6 where it can be appreciated that the maps focus on the given objects assigning low probabilities to points that lie in other regions.


(a) p^L and p^R

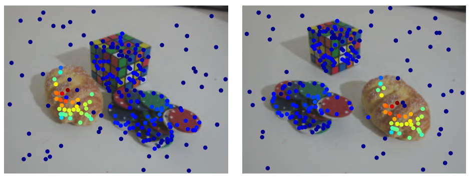
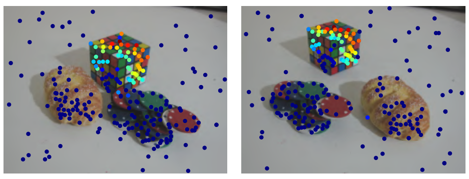

(b) ψ

Fig. 5.5: Comparison between disjoint and joint probabilities. (a) graphically shows the two disjoint discrete probabilities p^L and p^R; (b) graphically show the joint discrete probability ψ. Original figure from AdelaideRMF data-set [25].


(a)

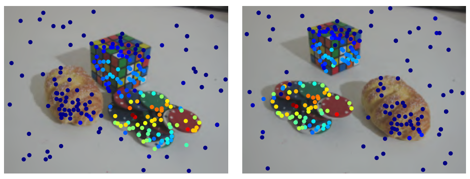

(b)

Fig. 5.6: (a) shows the semantic prior of the cube; (b) shows the semantic prior of the chips. Original figure from AdelaideRMF data-set [25].

Alg. 5 formally describes the process illustrated so far. Note, we assume to have already trained a CNN (as in Fig. 5.3) for the multi-label classification setting (a set of labels is returned by the network). The algorithm takes as input a couple of pictures $\left(I_L \in \mathbb{R}^{H\times W\times 3}, I_R \in \mathbb{R}^{H\times W\times 3}\right)$, as shown in Fig. 5.1a and a set of corresponding points $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \ldots, N\right\}$ where $\mathbf{x}_n^\square = \left(x_n^\square, y_n^\square,\right)$ is a point in image $I_\square$ and $\square \in \{L, R\}$. In line 1 and 2 we independently feed the CNN with I_L and I_R to get two sets of labels $\bar{\Lambda}^L$ and $\bar{\Lambda}^R$, one for each image. To reduce the error, in line 3 we compute the intersection of these sets and get $\bar{\Lambda}$. For each label (or class) $\lambda \in \bar{\Lambda}$, in lines 5 and 6 we compute the CAMs of I_L and I_R and we get M_λ^L and M_λ^R. Then, in line 9, for each point $\mathbf{x}_n^L$ in the left image, we will assign to p_n^L the value of M_λ^L at the coordinate of the point. In line 10, we do the same for all the points in the right image. Once we have the two probability vectors p^L and p^R, in line 11 we compute the element-wise minimum to get ψ_λ. We repeat the process for all the labels $\lambda \in \bar{\Lambda}$. In line 13, we store each ψ_λ in a set Ψ that will contain all the discrete probabilities for all the objects. We use the probabilities $\psi \in \Psi$ as priors for the sampling.

Algorithm 4: Semantic priors from CAMs

```
Input  : $\left(I_L \in \mathbb{R}^{H\times W\times 3}, I_R \in \mathbb{R}^{H\times W\times 3}\right)$ – image pair
         $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \dots, N\right\}$ – set of matches

Output : $\psi$ – semantic prior for class $\lambda$

1  $\bar{\Lambda}^L \subseteq \Lambda \leftarrow$ Retrieve the classes image $I_L$ belong to
2  $\bar{\Lambda}^R \subseteq \Lambda \leftarrow$ Retrieve the classes image $I_R$ belong to
3  $\bar{\Lambda} = \bar{\Lambda}^L \cap \bar{\Lambda}^R$
4  for $\lambda \in \bar{\Lambda}$ do
5      $M_\lambda^L \leftarrow$ Compute the CAM of $I_L$ for class $\lambda$
6      $M_\lambda^R \leftarrow$ Compute the CAM of $I_R$ for class $\lambda$
7      $\psi_\lambda \leftarrow$ Initialize array of length $N$
8      for $n \in \{1, \dots, N\}$ do
9          $p_n^L = M_\lambda^L(\mathbf{x}_n^L)$
10         $p_n^R = M_\lambda^R(\mathbf{x}_n^R)$
11         $\psi_{\lambda,n} \leftarrow \min\left\{p_n^L, p_n^R\right\}$
12     end
13     $\Psi \leftarrow$ Add $\psi_\lambda$ to set $\Psi$
14 end
15 return $\Psi$
```

5.1.2 Semantic prior from features

In the previous subsection (Sec. 5.1.1), we described how to extract the semantic prior from CAMs. In this subsection, we will see that we are not forced to use CAMs, we can directly exploit the feature maps. In practice, we move towards a Transfer Learning framework: in the source task S we train a CNN for (multi-class or multi-label) image-classification using a data-set TR^S, then we remove the classifier (as in Fig. 5.7) and we use the features as semantics for the sampling. In this case the sampling is the destination task and, unlike standard Transfer Learning, we do not have a data-set TR^D and we do not even know the object-labels the network is going to see, that is why it makes no sense to design a new classifier on top of the feature extractor. Note, we rely on the assumption that the network has been trained on a wide enough data-set so that a good range of object-patterns can be localized, including new patterns that are somehow similar to patterns already learned from the network.

Alg. 5 formally describes how to extract a set Ψ of discrete probabilities that we use as priors for sampling. The algorithm takes as input a couple

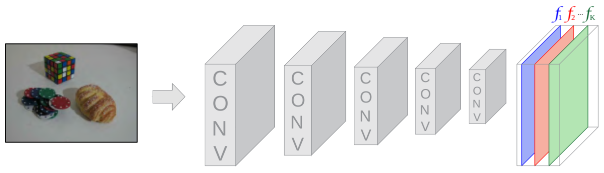


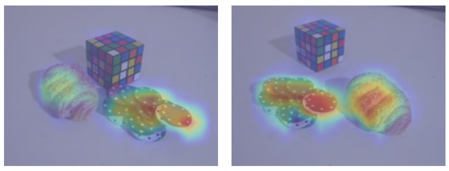
Fig. 5.7: CNN architectures to extract semantics by using the features from the last convolutional layer. Original figure from [32].

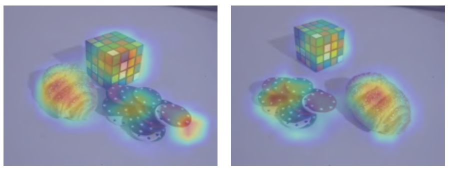
pictures $\left(I_L \in \mathbb{R}^{H\times W\times 3}, I_R \in \mathbb{R}^{H\times W\times 3}\right)$, as shown in Fig. 5.1a and a set of corresponding points $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \dots, N\right\}$ where $\mathbf{x}_n^\square = \left(x_n^\square, y_n^\square,\right)$ is a point in image $I_\square$ and $\square \in \{L, R\}$. In lines 1 and 2 we separately feed the FE (as shown in Fig. 5.7) with I_L and I_R. From $I_\square$ we extract K feature maps $\bar{f}_k^\square \in \mathbb{R}^{\bar{H}\times\bar{W}}$. Since we need to overlap $I_\square \in \mathbb{R}^{H\times W\times 3}$ and $\bar{f}_k^\square \in \mathbb{R}^{\bar{H}\times\bar{W}}$, in line 4 and 5 we resize $\bar{f}_k^\square$ so that they are the same spatial size and we get $f_k^\square \in \mathbb{R}^{H\times W}$. We rearrange the feature maps as couples $\left(f_k^L, f_k^R\right)$, as shown in Fig. 5.8. Each couple highlight the same patterns in the left and right image respectively. Let us focus on a couple of feature maps $\left(f_k^L, f_k^R\right)$ and match $\left(\mathbf{x}_n^L, \mathbf{x}_n^R\right)$. The value $f_k^\square(\mathbf{x}^\square)$ represents the probability that point $\mathbf{x}^\square$ belongs to an image-object according to feature map $f_k^\square$. A match is likely to relate two points of the same geometric model if both of them have high probability in their respective feature map. Therefore, in line 7 we compute the probability of the match as the minimum of the probabilities of its endpoints:

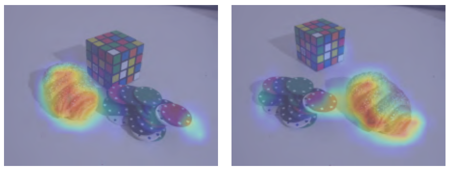
$$\psi_{k,n} = \min\left\{f_k^L(\mathbf{x}_n^L), f_k^R(\mathbf{x}_n^R)\right\} \tag{5.5}$$

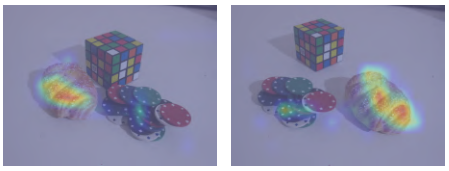
This way, if both points are object-points, they will have a high value on their feature map. If one of them is an outlier (or pseudo-outlier), it will have a low probability, and by computing the minimum we assign low probability to the match. Note, in order to compute the semantic prior, we are jointly using the semantics of both the left and right image. Repeating the computation for all the N matches and all the K feature maps, we get K discrete probabilities ψ_k that are added to set Ψ in line 9. Fig. 5.9 shown some of this features: compared to the CAMs that localize a single object-pattern, feature maps can highlights a sub-region of the pattern or even part of the background but, as soon as the CNN is correctly trained, it should focus on image objects rather than background.

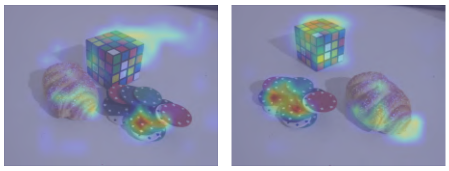
As a final remark, let's discuss the value of K, i.e., the number of filters in the last convolutional layer. We have to set a small value for K: this allows us to reduce the number of semantic priors to be considered during the sampling and to compress the dimensionality of the latent representation. We set $K = C$, where C is the number of classes. This choice did not harm the

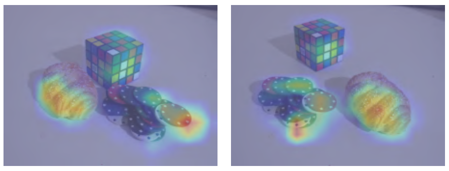

(a)

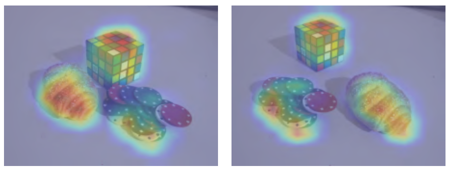

(b)

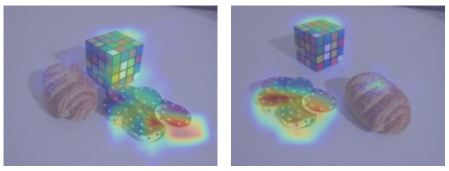

(c)

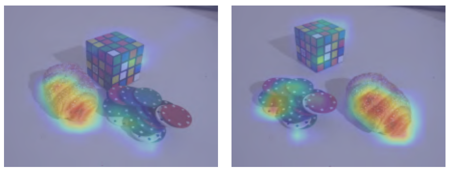

(d)


(e)


(f)


(g)


(h)


(i)

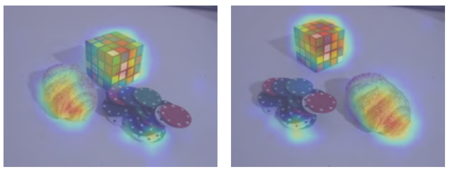
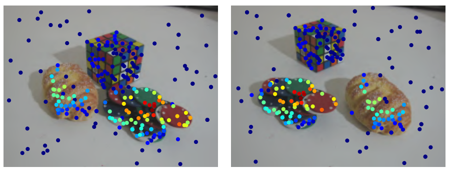
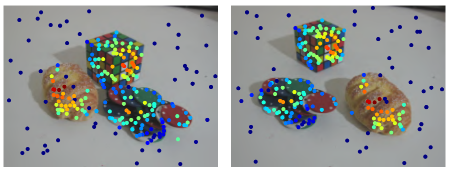
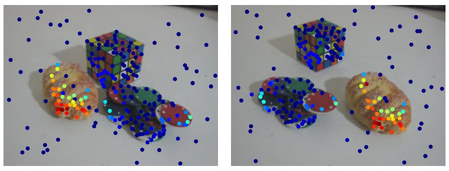
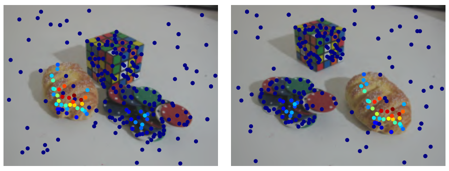

(j)

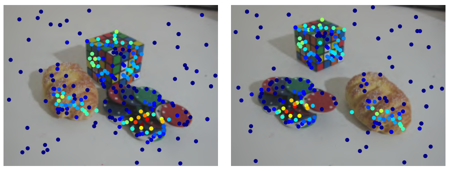
Fig. 5.8: Feature maps of breacubechips. Each feature map highlights a particular pattern or shape. Original figure from AdelaideRMF data-set [25].

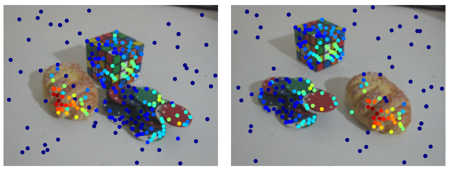

(a)

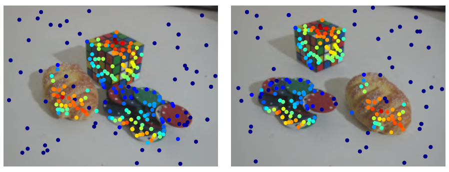

(b)

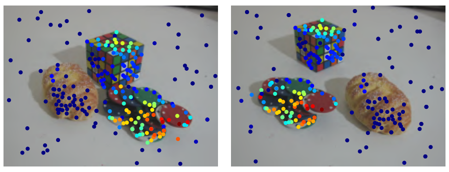

(c)

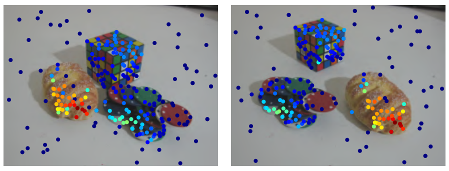

(d)


(e)


(f)


(g)


(h)


(i)

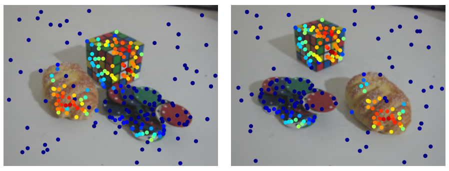

(j)

Fig. 5.9: Some semantic priors from features. Original figure from AdelaideRMF data-set [25].

classification accuracy in the source problem. In the next section we describe how to use the set of discrete probabilities as priors for the sampling phase.

```
Algorithm 5: Semantic priors from features
  Input  : $\left(I_L \in \mathbb{R}^{H \times W \times 3}, I_R \in \mathbb{R}^{H \times W \times 3}\right)$ − image pair
           $FE$ − CNN feature extractor
           $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \ldots, N\right\}$ − set of matches

  Output : $\Psi = \left\{\psi_k \in \mathbb{R}^{|P|} \mid k = 1, \ldots, K\right\}$ − semantic priors

1  $\bar{f}_1^L, ..., \bar{f}_K^L \leftarrow FE\,(I_L)$
2  $\bar{f}_1^R, ..., \bar{f}_K^R \leftarrow FE\,(I_R)$
3  for $k \in \{1, \ldots, K\}$ do
4      Spatially resize $\bar{f}_k^L \in \mathbb{R}^{\bar{H} \times \bar{W}}$ to get $f_k^L \in \mathbb{R}^{H \times W}$
5      Spatially resize $\bar{f}_k^R \in \mathbb{R}^{\bar{H} \times \bar{W}}$ to get $f_k^R \in \mathbb{R}^{H \times W}$
6      for $n \in \{1, \ldots, N\}$ do
7          $\psi_{k,n} \leftarrow \min\left\{f_k^L(x_n^L, y_n^L), f_k^R(x_n^R, y_n^R)\right\}$
8      end
9      $\Psi \leftarrow$ Add $\psi_k$ to set $\Psi$
10 end
11 return $\Psi$
```

5.2 Semantic-aware sampling

So far we have seen that robust estimators rely on a sampling phase to select a set of models explaining the data points. This mechanism allows us to be robust to outliers since we do not take into account all the points at once. In this section, we present a novel sampling strategy called *semantic-aware sampling* that exploits the semantic priors $\Psi = \{\psi_k \mid k = 1, \ldots, K\}$ of an image pair to guide the sampling phase towards models that can better explain the data. The algorithm is formally described in Alg. 6. It takes as input an image pair (I_L, I_R), a set of corresponding points $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \ldots, N\right\}$, a scalar s representing the number of samples to be drawn and a parameter ν who regulates the ray of the locality circumference.

Algorithm 6: Semantic-aware Sampling

```
Input  : $\left(I_L \in \mathbb{R}^{H\times W\times 3}, I_R \in \mathbb{R}^{H\times W\times 3}\right)$ − image pair
         $\left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \dots, N\right\}$ − set of matches
         s − number of samples required to uniquely fit the model
         $\nu$ − parameter to regulate the locality prior

Output : $\Omega$ − preference matrix

1  $\Psi \leftarrow f_{\text{semantic prior}}\left((I_L, I_R), \left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n \in \{1, \dots, N\}\right\}\right)$
2  $T \leftarrow 3N$
3  while $t < T$ do
4      $k \leftarrow \left\lfloor \frac{t}{N} \right\rfloor \mod N$
5      MSS $\leftarrow f_{\text{MSS sampling}}\left(\left\{\mathbf{x}_n^L, \mathbf{x}_n^R \mid n = 1, \dots, N\right\}, \psi_k, \text{s}, \nu\right)$
6      $F \leftarrow f_{\text{model}}(\text{MSS})$
7      $r \leftarrow f_{\text{residual}}\left(F, \left\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \dots, N\right\}\right)$
8      for $n \in \{1, \dots, N\}$ do
9          if $r_n < \tau$ then
10             $col_n \leftarrow e^{-\frac{r}{\tau}}$
11         end
12         else
13             $col_n \leftarrow 0$
14         end
15     end
16     $\Omega \leftarrow$ Add column $col_n$
17 end
18 return $\Omega$
```

In line 1 we extract the set Ψ of semantic priors that we use in line 5 to sample the minimal sample sets, the procedure is shown in Alg. 7. In line 6 we fit a geometric model and in line 7 we compute the residuals. From line 8 to line 14 we apply Eq. 2.4 that for convenience we report below:

$$\Omega(i,j) = \begin{cases} e^{-\frac{r(i,j)}{\tau}} & \text{if } r(i,j) \leq 5\tau \\ 0 & \text{otherwise} \end{cases}$$

where i is a point and j is a geometric model. In line 16, we add a new column to the preference matrix Ω by using the vector previous computed. At the end, the algorithm returns the preference matrix Ω (line 18).

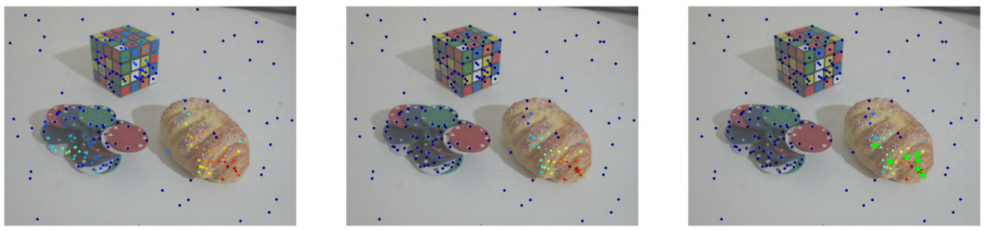
Algorithm 7: MSS sampling using semantic prior

```
Input  : $\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \ldots, N\}$ − set of matches
         $\psi_k$ − prior probability vector
         s − number of samples required to fit the model
         $\nu$ − parameter to regulate the locality prior

Output : MSS − minimal sample set

1  $\psi_k \leftarrow \frac{\psi_k}{\sum_{n=1}^{N} \psi_{k,m}}$
2  $< \mathbf{x}^L, \mathbf{x}^R > \leftarrow f_{\text{sample}}\left(\{< \mathbf{x}_n^L, \mathbf{x}_n^R > \mid n = 1, \ldots, N\}, 1, \psi_k\right)$
3  $\bar{\psi}^L \leftarrow f_{\text{locality}}\left(\{\mathbf{x}_n^L \mid n = 1, \ldots, N\}, \mathbf{x}^L, \nu\right)$
4  $\bar{\psi}^R \leftarrow f_{\text{locality}}\left(\{\mathbf{x}_n^R \mid n = 1, \ldots, N\}, \mathbf{x}^R, \nu\right)$
5  $\bar{\psi} \leftarrow$ Compute the element-wise min between $\bar{\psi}^L$ and $\bar{\psi}^R$
6  $\bar{\psi} \leftarrow \bar{\psi} \odot \psi_k$
7  $\bar{\psi} \leftarrow \frac{\bar{\psi}}{\sum_{n=1}^{N} \bar{\psi}_n}$
8  MSS $\leftarrow f_{\text{sample}}\left(\{\mathbf{x}_n^L \mid n = 1, \ldots, N\} \setminus \mathbf{x}, s - 1, \bar{\psi}\right)$
9  MSS $\leftarrow \{\mathbf{x}\} +$ MSS
10 return MSS
```

Let's now describe Alg. 7. In line 1 we normalize ψ_k as in Eq. 5.6:

$$\psi_k = \frac{\psi_k}{\sum_{n=1}^{N} \psi_k} \tag{5.6}$$

and we use it as prior to sample a single match $< \mathbf{x}^L, \mathbf{x}^R >$ (line 2). Since closer points are more likely to belong to the same geometric model [15], in lines 3 and 4 we compute the locality priors $\bar{\psi}^L$ and $\bar{\psi}^R$ for the match end-points where probability $\bar{\psi}_n^{\square}$ for $n \in \{1, \ldots, N\}$ depends on the Euclidean distance from $\mathbf{x}^{\square}$ with $\square \in \{L, R\}$. In line 5 we aggregate $\bar{\psi}^L$ and $\bar{\psi}^R$ into a single discrete probability vector $\bar{\psi}$ for the match by using the element-wise min. In line 6 we compute the Hadamard product of ψ_k and $\bar{\psi}$ and we use



(a) (b) (c)

Fig. 5.10: Combination of semantics and locality. Fig. (a) shows the point semantics extracted from the features: the points on the bread are the most likely to be sampled but also some points on the chips have non-zero probability to be sampled. Fig. (b) shows the effect of combining semantics and locality: a point on the bread is sampled, so points on the cube and chips have a very low probability of being sampled. Note that, by using only the localized prior, we would have sampled background points. Fig. (c) shows the points that are sampled in this iteration. Original figure from AdelaideRMF data-set [25].

this prior to sample the remaining $s-1$ points (line 8). The rationale is that we want to sample both image-object and close points: ψ_k tells us which are the image-object points and $\bar{\psi}$ filters points far away from the first sampled one. This allows to consider a wider area (larger ν) around the object without the risk to sample outliers. This reasoning is graphically shown in Fig. 5.10: we sample the first point by using the semantics in Fig. 5.10a, then we apply the locality and we get the probability distribution in Fig. 5.10b which is then used to sample the other seven points as shown in Fig. 5.10c. At the end, the algorithm returns the MSS (line 10).

Algorithm 8: Compute locality prior

```
Input  : $\{\mathbf{x}_n \mid n = 1, \dots, N\}$ – set of data points
         $\mathbf{x}_\alpha = (x_\alpha, y_\alpha)$ – coordinates of the just sampled point
Output : $\bar{\psi}$ – discrete probability vector

1 for $n \in \{1, \dots, N\}$ do
2     $(x_n, y_n) = \mathbf{x}_n$
3     $d^2 = (x_n - x_\alpha)^2 + (y_n - y_\alpha)^2$
4     $\sigma = \nu d$
5     $\bar{\psi}_m = e^{-\frac{d^2}{\sigma^2}}$
6 end
7 return $\bar{\psi}$
```

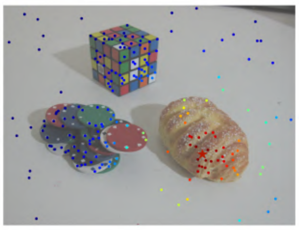
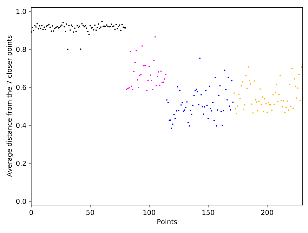


Fig. 5.11: Locality example. When we sample a point $\mathbf{x}_\alpha$, denoted with the red star, the probability of the other points depends on their distance from $\mathbf{x}_\alpha$. Original figure from AdelaideRMF data-set [25].

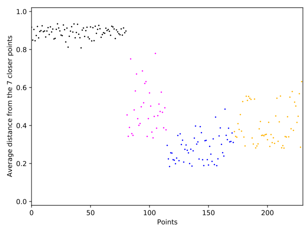
The sampling is computed using function f_{sample} whose parameters are respectively a *population* of points, the *number of points* to be sampled and the prior discrete probability over the points. The sampling is always without replacement. The locality prior is computed using function f_{locality} whose parameters are respectively a *population* of points, a *reference point* $\mathbf{x}_\alpha$ against which to compute the distance and a parameter ν who balances the relationship between semantics and distance. It is chosen heuristically. The function f_{locality} is described in Alg. 8. In line 2 we just unpack the coordinates of the n-th point (done just to make the next step more clear). In line 3 we compute the squared distance of the n-th point form $\mathbf{x}_\alpha$. In line 4 we compute σ as the multiplication of the distance and the parameter ν who regulates the notion of proximity to the point $\mathbf{x}_\alpha$. Finally, in line 5 we compute the discrete probability of the points: the probability is inversely proportional to the distance from $\mathbf{x}_\alpha$, the smaller the distance, the greater the probability of sampling the point. The result of this process is shown in Fig. 5.11 where the red star represents $\mathbf{x}_\alpha$ and its neighbors have larger probability to be sampled than far away points. Note that, since there is no semantics, we also give high probability to background points. The algorithm returns the discrete probability vector $\bar{\psi}$.

5.3 ERGO

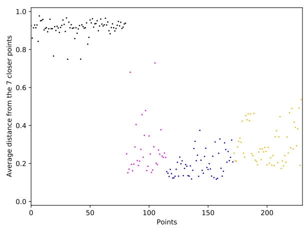
Our semantic-aware sampling allows to drive the sampling towards the foreground and, being based on semantics, can also separate points of different object being very effective in the multi-model case, especially for preference-base algorithms that rely on a good representation of the points in the preference space. In this section we introduce ERGO, a method that allows us to filter out gross-outliers before the model estimation. Recall that gross-outliers are those points not belonging to any geometric model and that



(a) Uniform

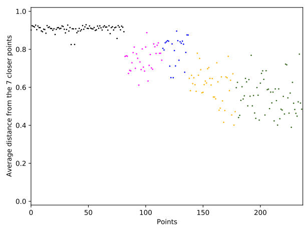


(b) Localized

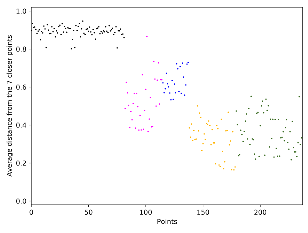


(c) Semantic-aware

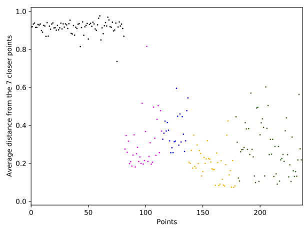
Fig. 5.12: ERGO plots for breadcubechips. The plots have on x-axis the ordinal value of the points and on the y-axis the average Tanimoto distance w.r.t. its $s - 1$ closest points, where s is the cardinally of the MSS. For graphical reasons, points belonging to the same clusters have the same color and are ordered. Black points are gross-outliers. (a) comes from a uniform sampling; (b) comes from a localized sampling; (c) comes from a semantic-aware sampling.



(a) Uniform



(b) Localized



(c) Semantic-aware

Fig. 5.13: ERGO plots for breadcartoychips. The plots have on x-axis the ordinal value of the points and on the y-axis the average Tanimoto distance w.r.t. its $s - 1$ closest points, where s is the cardinally of the MSS. For graphical reasons, points belonging to the same clusters have the same color and are ordered. Black points are gross-outliers. (a) comes from a uniform sampling; (b) comes from a localized sampling; (c) comes from a semantic-aware sampling.

arise due to background clutter or measurement errors. As we have seen in the Sec. 2.1, preference-based algorithms cluster the points in the preference space because points belonging to the same geometric model appears as dense regions while gross-outliers are sparse points in this space.

In Fig. 5.12 and Fig. 5.13 we represent points based on their neighborhood in the preference space: for each point we compute the average Tanimoto distance from its C neighbors thus accessing the density of points around it. Estimating the right value of C is not an easy task. On the one hand, the more points we consider, the more confident we can be that the point is an inlier to some model and does not belong to a set of outliers that by chance are close together in the preference space. On the other hand, considering too many points would be harmful because we could consider points belonging to different models and the average distance would no longer be relevant. In general, we should set C to the minimum cluster cardinality expected for the models. In the worst case, to be sure we do not take points from other models, we can consider $C = s - 1$, where s is the cardinality of the minimal sample set (MSS). The rationale behind this choice is that a cluster corresponding to a model must contain at least s points otherwise we would never be able to sample a pure MSS and the cluster would be considered too small and therefore discarded.

In Fig. 5.12 and Fig. 5.13 we use the representation of the points based on the neighborhood to compare uniform, localized and semantic-aware sampling. First, we observe that by using localized sampling rather than uniform sampling, the points belonging to some model (colored pink, blue, orange and green) drop towards zero. This effect is emphasized when using semantic-aware sampling. Moreover, inspecting the plots we see that gross-outliers (colored black) are all close to one. Recall that gross-outliers are points not belonging to any model, retrieved due to background clutter or errors in the point-matching phase, and which we would like to discard.

When we focus on Fig. 5.12c and Fig. 5.13c, both produced by using the semantic-aware sampling, we notice that the gap between inliers to some model (colored pink, blue, orange and green) and gross-outliers (colored black) increase therefore making less severe the choice of a threshold to filter out outliers. In other words, we propose to filter out points that in the Tanimoto space have average distance to the neighborhood larger than a threshold. Usually the outlier rejection phase is performed using robust statistical techniques only after an estimate of the geometric models has already been computed. Our sampling strategy allows us to improve the preference embedding so that we can perform outlier rejection at an early stage, when models have not yet been fitted. We called this thresholding technique in the preference space *Early Rejection of Gross-Outliers* (ERGO for short).

Early rejection may occasionally filter out inliers sparse on the preference space. When the number of inliers lost due to ERGO is small, we can retrieve these points in a subsequent refitting step, this is a common practice in robust estimation. For example this is the case of Fig. 5.13c: setting the threshold to 0.6 we lose two points. On the other hand, if the number of lost inliers is large, we may no longer be able to fit the corresponding models. For example this is the case of Fig. 5.12c: setting the threshold to 0.6 we lose all the points in the pink and blue cluster and almost half of the points in the orange and green clusters. That is the reason why ERGO is so effective when associated to a good sampling strategy that sharply separate inliers and outliers.

Algorithm 9: ERGO

```
Input  : $X$ − set of $N$ points described in the preference space
         $C$ − number of neighbors to consider
         $\delta$ − threshold to reject points
Output : $X'$ − set of good points $\lambda$

1  $X' = \emptyset$
2  for $i \in \{1, \dots, N\}$ do
3      $\xi \leftarrow$ Initialize array of length $N - 1$
4      for $j \in \{1, \dots, N\} \setminus i$ do
5          $d_{ij} \leftarrow$ Compute the Tanimoto distance between $x_i$ and $x_j$
6          $\xi \leftarrow$ Add $d_{ij}$ to set $\xi$
7      end
8      $\xi' \leftarrow$ Sort $\xi$
9      $\hat{d} \leftarrow \frac{1}{c} \sum_{c=1}^{C} \xi'_c$
10     if $\hat{d} \leq \delta$ then
11         $X' \leftarrow$ Add point $x_i$ to $X'$
12     end
13 end
14 return $X'$
```

ERGO is formally described in Alg. 9. It takes as input the set X of data points represented by their preference functions (or preference sets), the number C of neighboring points we are going to consider and a threshold δ. In line 1 we initialize the set X' of *good points* that will be returned by the algorithm. By *good points* we mean the points that are inliers to some geometric model. In order to compare all the pairs of points we have implemented two nested loops in lines 2 and 4 both iterating over the data points. Note that the second loop excludes x_i to avoid the comparison of a point with itself. Given the point x_i, in line 3 we initialize an array of length $N - 1$, since for each point we compute $N - 1$ distances. From line 4 to line 7, we compute the average Tanimoto distance $\hat{d}$ between x_i and the C closest points in the preference space. Finally, from line 10 to line 12 we check if the average distance between x_i and its neighborhood is below the threshold and eventually we add the point to the set X'.

Summing up, using a semantic-aware sampling allows for more sharp separation of inliers and outliers, makes a thresholding technique like ERGO less critical and reduce the number of inliers we lose in the thresholding.

5.4 Summary

This chapter has formally presented the *semantic-aware sampling*, a sampling strategy that can be embedded in robust estimation to achieve smaller misclassification error while reducing the number of samplings. The reason behind these results is the better representation of the points in the preference space given by our sampling strategy.

Let's recap the flow of computation. Given a couple (or more) pictures representing the same dynamic scene, in order to extract the geometric models, we follow two paths. On the one hand we extract the SIFT keypoints and descriptors and we match them in the descriptor space to find a set of corresponding points. On the other hand, we use a CNN to extract the semantics (features or CAMs). Then, for each point that has coordinate (x, y) in the ambient space, we associate a probability by using the semantics. Finally, we assign each match the minimum probability of its endpoints. This way we obtain a discrete probability vector. At the end of the process, we have a set of discrete probabilities that can be used as priors for the sampling phase of the robust estimator. Optionally, we can use ERGO to perform an early rejection of gross outliers by using just the preference matrix. This simple yet powerful idea will be test in the next chapter devoted to experiments.

Chapter 6

Experiments

In this chapter we present the experiments carried out to validate our method. Sec. 6.1 describes the setting we used for the experiments and Sec. 6.2 reports the results of the comparison of our approach with other methods.

6.1 Setting

6.1.1 Test data-set

For testing we used the standard AdelaideRMF [25], a data-set for robust geometric model fitting (homography estimation and fundamental matrix estimation). It is composed of 38 image pairs (19 for motion segmentation and 19 for plane segmentation) with matching points corrupted by gross-outliers. The ground-truth segmentations are also available.

6.1.2 Evaluation metric

In our experiments we took into account two measures. First, the number Υ of pure MSS sampled by the algorithm. Suppose we perform M samplings, we compute the number of pure MSS as

$$\Upsilon = \sum_{i=0}^{M-1} \upsilon_i \tag{6.1}$$

where υ_i measure if all the points belongs to the same geometric model according to the ground-truth:

$$\upsilon_i = \begin{cases} 1 & \text{if points in MSS all belong to the same model} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

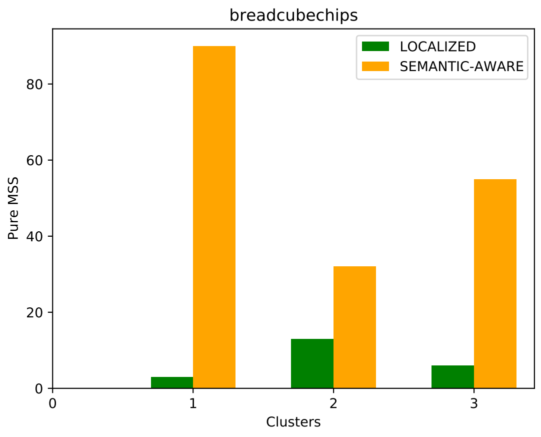


Fig. 6.1: Pure MSS histogram for breadcubechips. The x-axis represents the clusters (1 is “bread”, 2 is “cube”, 3 is “chips”), the y-axis represents the number of pure MSS for each cluster. Green denotes the localized sampling strategy, while yellow denotes the semantic-aware sampling strategy.

We also used a fine-grained counter to measure the number of pure MSS for each geometric model. An example is shown in Fig. 6.1.

Moreover, we are interested to see how the sampling improve the ability of the algorithms to find the correct geometric models in the data, thus we adopted the standard evaluation metric for robust multi-model fitting, namely the Misclassification Error (ME), defined as:

$$ME = \frac{\text{number of misclassified points}}{\text{number of points}} \tag{6.3}$$

where a point is *misclassified* when it is assigned to a wrong geometric model according to the ground-truth. For a better reading, in the following, we will refer to the percentage misclassification error, denoted as ME %. Note that, since there is no one-to-one mapping between the ground-truth and the estimated mask, in order to check if a point is misclassified, we need to consider all possible label-mappings between the two masks and select the one that minimizes the error.

6.1.3 Base network and training data-set

For the experiments, we selected Inception V3 [31] as base network. It is the third edition of Google’s Inception Convolutional Neural Network [28], originally introduced during the ImageNet Recognition Challenge [20]. We cut the network immediately after layer 279 (mixed 9) and we performed

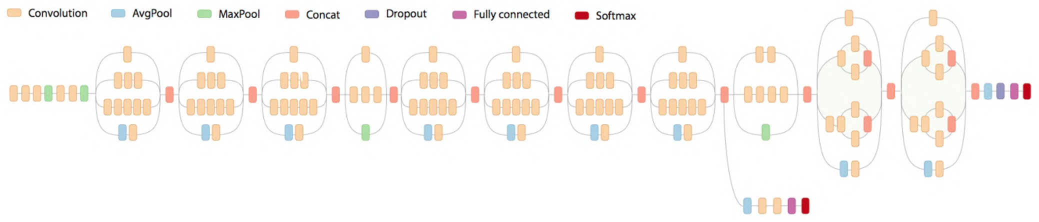


Fig. 6.2: Inception V3 Architecture.

fine-tuning by using a model pre-trained on ImageNet and freezing the layers up to 228. We used this configuration because it was the best one after performing hyper-parameter tuning.

We trained the Convolutional Neural Networks previously described on two data-set. For motion segmentation task (fundamental matrix estimation) we used a data-set of common objects. We have selected the *Caltech 256* [18], a data-set of 256 object categories containing a total of 30607 images. It is an improvement to its predecessor, the Caltech 101 data-set [14], which is a standard data-set for classification and localization. Caltech 256 was collected by downloading examples from Google Images and then manually grouping the images by category. For plane segmentation (homography estimation) we used a data-set representing buildings. In particular, we have selected the Architectural Style data-set [29] designed to recognize the architectural style of a building in an image. The rationale behind this choice is that, since we cannot distinguish between planes using this CNN architecture, it is enough to distinguish between foreground and background. This is the reason why we will see that the results for plane segmentation will not exceed those of the competitors.

6.1.4 Tools and libraries

We implemented the proposed method using *Python 3.6.9.* To support the implementation of the algorithm and visualization of the results we have used many libraries, the main ones are listed below.

- **Numpy.** It is a Python library that provides a multidimensional array objects, various derived objects, and an assortment of routines for fast operations on arrays [47]. We extensively used this package to store all the information (usually tensors) and for efficiency we exploited its built-in sampling functions.

- **Tensorflow.** It is a free and open-source software library developed by Google for machine learning. It has a particular focus on training and inference of deep neural networks [33]. We used this framework to train all the Convolutional Neural Networks used in the experiments and to extract feature maps and CAMs from images.

- **OpenCV.** Open Source Computer Vision Library is an open-source computer vision and machine learning software library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products [48]. We have extensively used the RANSAC implementation for homography and fundamental matrix estimation and it has been very useful in general for image manipulation.

- **Pandas.** It is a software library for data manipulation and analysis [54]. In particular, it offers data structures and operations for manipulating numerical tables and time series. We mainly used it for data manipulation.

- **Matplotlib.** It is a comprehensive library for creating static, animated, and interactive visualizations in Python [45]. We used this library to show all the pictures and plots in this work and throughout the all experimental phase.

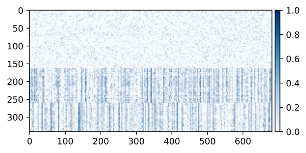
6.2 Experimental results

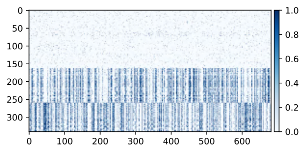
6.2.1 Semantic-aware sampling with T-Linkage

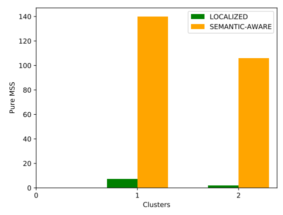
We approached the experimental stage using T-Linkage as a reference algorithm. We implemented it from scratch in Python, tuning the thresholds so that the error in the best run was minimized[1]. We then replaced the localized sampling with semantic-aware sampling and repeated the experiments. The results of some experiments are graphically shown in Fig. 6.3, Fig. 6.4, Fig. 6.5, Fig. 6.6 and Fig. 6.7.

A complete overview of the results is shown in Tab. 6.1. The table shows the percentage misclassification error of T-Linkage equipped respectively with localized sampling in the first column and with semantic-aware sampling in the second column. Below is the mean and the median of the values in the two columns. Note, to avoid fluctuations due to particularly lucky or unlucky runs, the results are the average of 100 iterations. From the table we see that the results of our method are significantly better.

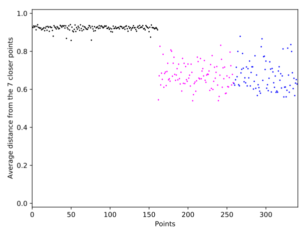
[1]In the experiments, we consider the average misclassification error for 100 runs, not the minimum.


(a) Localized

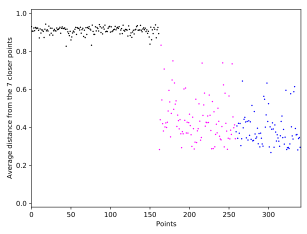

(b) Semantic-aware



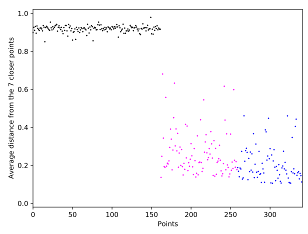
(c) Pure MSS



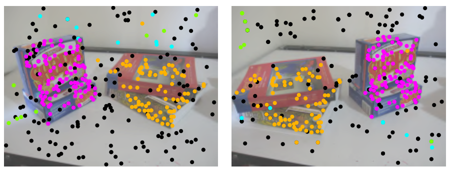
(d) Uniform



(e) Localized



(f) Semantic-aware


(g) Localized - ME % = 5.7915

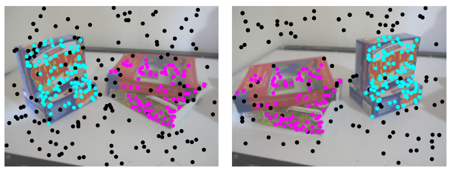
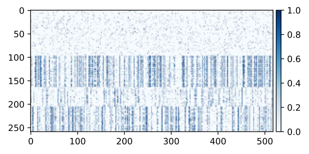
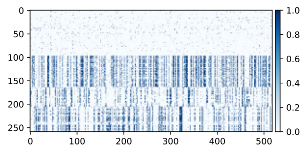
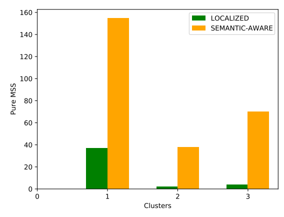

(h) Semantic-aware - ME % = 0.0

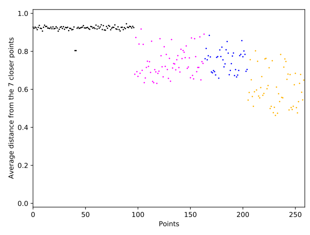
Fig. 6.3: Experiment: biscuitbook. (a) and (b) show the preference matrix with the localized and semantic-aware sampling respectively. (c) show the histogram of Pure MSS for each cluster (1 is “biscuit”, 2 is “book”). Green and yellow denote the localized and semantic-aware sampling, respectively. (g) and (h) show the cluster masks for localized and semantic-aware sampling with the corresponding percentage Misclassification Error. (e) and (f) plot the points based on their average distance from the neighborhood. Note, the semantic-aware sampling pushes inliers towards zero and outliers towards one. Original figure from AdelaideRMF data-set [25].


(a) Localized

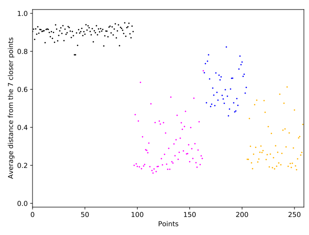

(b) Semantic-aware



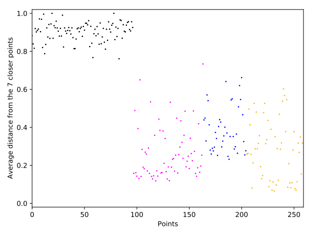
(c) Pure MSS



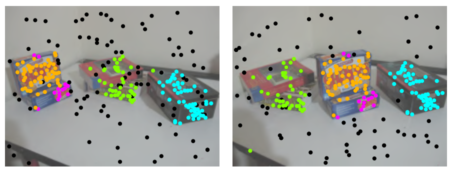
(d) Uniform



(e) Localized



(f) Semantic-aware


(g) Localized - ME % = 6.4516

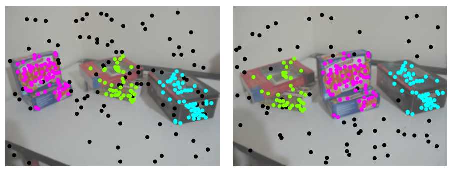
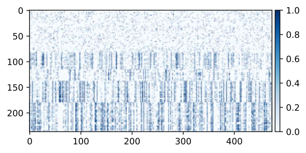
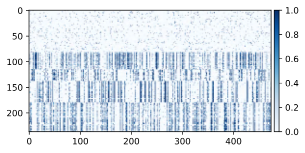

(h) Semantic-aware - ME % = 0.0

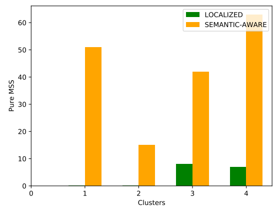
Fig. 6.4: Experiment: biscuitbookbox. (a) and (b) show the preference matrix with the localized and semantic-aware sampling respectively. (c) show the histogram of Pure MSS for each cluster (1 is “biscuit”, 2 is “book”, 3 is “box”). Green and yellow denote the localized and semantic-aware sampling, respectively. (g) and (h) show the cluster masks for localized and semantic-aware sampling with the corresponding percentage Misclassification Error. (e) and (f) plot the points based on their average distance from the neighborhood. Note, the semantic-aware sampling pushes inliers towards zero and outliers towards one. Original figure from AdelaideRMF data-set [25].



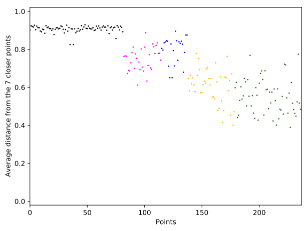
(a) Localized



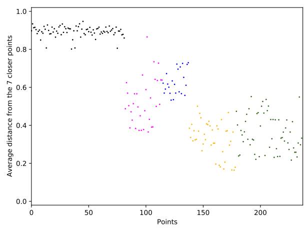
(b) Semantic-aware



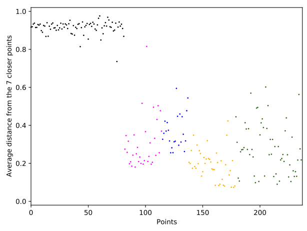
(c) Pure MSS



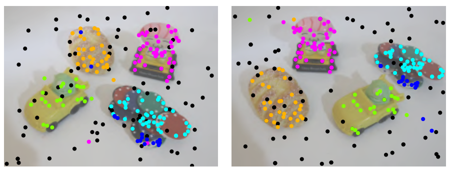
(d) Uniform



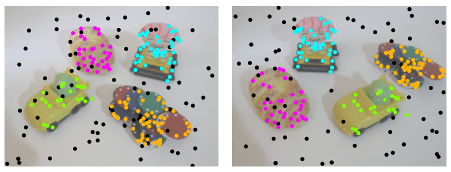
(e) Localized



(f) Semantic-aware

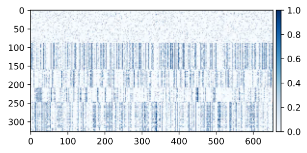


(g) Localized - ME % = 10.9705

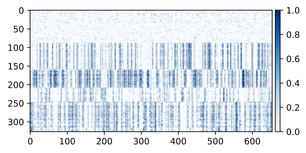


(h) Semantic-aware - ME % = 0.8439

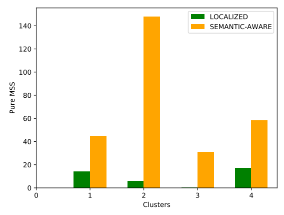
Fig. 6.5: Experiment: breadcartoychips. (a) and (b) show the preference matrix with the localized and semantic-aware sampling respectively. (c) show the histogram of Pure MSS for each cluster (1 is "bread", 2 is "car", 3 is "toy", 4 is "chips"). Green and yellow denote the localized and semantic-aware sampling, respectively. (g) and (h) show the cluster masks for localized and semantic-aware sampling with the corresponding percentage Misclassification Error. (e) and (f) plot the points based on their average distance from the neighborhood. Note, the semantic-aware sampling pushes inliers towards zero and outliers towards one. Original figure from AdelaideRMF data-set [25].



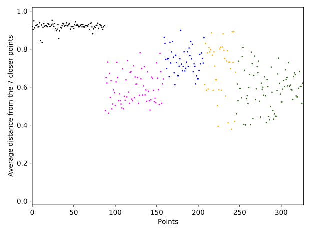
(a) Localized



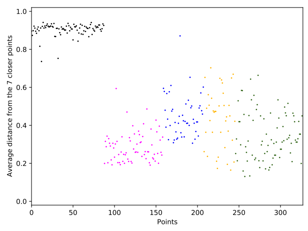
(b) Semantic-aware



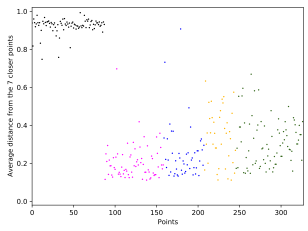
(c) Pure MSS



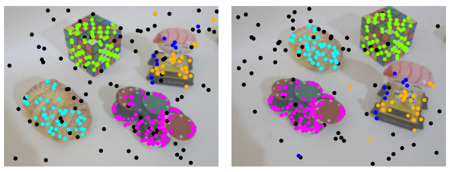
(d) Uniform



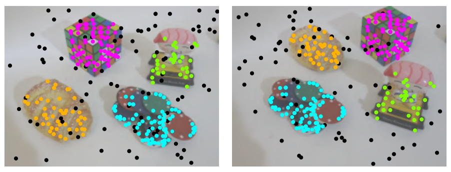
(e) Localized



(f) Semantic-aware

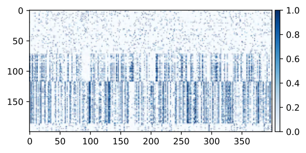


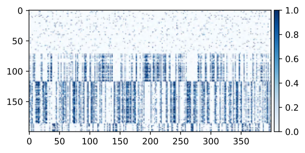
(g) Localized - ME % = 5.5046

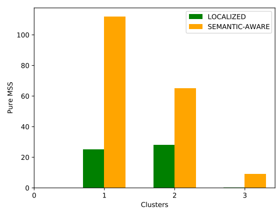


(h) Semantic-aware - ME % = 0.9174

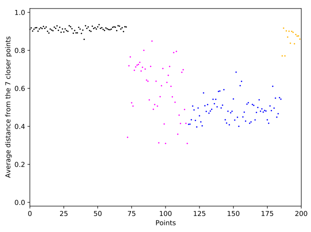
Fig. 6.6: Experiment: cubebreadtoychips. (a) and (b) show the preference matrix with the localized and semantic-aware sampling respectively. (c) show the histogram of Pure MSS for each cluster (1 is "cube", 2 is "bread", 3 is "toy", 4 is "chips"). Green and yellow denote the localized and semantic-aware sampling, respectively. (g) and (h) show the cluster masks for localized and semantic-aware sampling with the corresponding percentage Misclassification Error. (e) and (f) plot the points based on their average distance from the neighborhood. Note, the semantic-aware sampling pushes inliers towards zero and outliers towards one. Original figure from AdelaideRMF data-set [25].

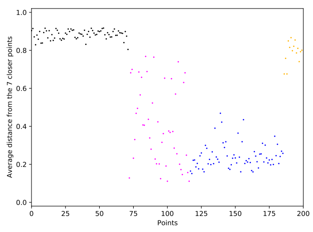

(a) Localized

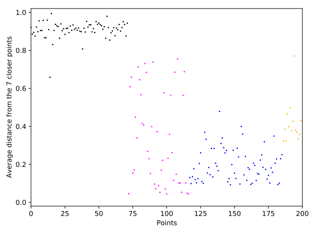

(b) Semantic-aware

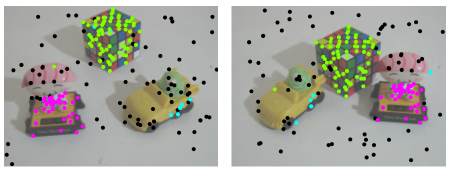


(c) Pure MSS


(d) Uniform


(e) Localized


(f) Semantic-aware


(g) Localized - ME % = 13.0

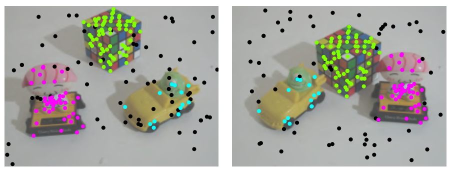

(h) Semantic-aware - ME % = 0.0

Fig. 6.7: Experiment: toycubecar. (a) and (b) show the preference matrix with the localized and semantic-aware sampling respectively. (c) show the histogram of Pure MSS for each cluster (1 is "toy", 2 is "cube", 3 is "car"). Green and yellow denote the localized and semantic-aware sampling, respectively. (g) and (h) show the cluster masks for localized and semantic-aware sampling with the corresponding percentage Misclassification Error. (e) and (f) plot the points based on their average distance from the neighborhood. Note, the semantic-aware sampling pushes inliers towards zero and outliers towards one. Original figure from AdelaideRMF data-set [25].

Index	Label	Localized	Semantic-aware
0	biscuit	4.9697	0.0152
1	biscuitbook	1.6158	0.0000
2	biscuitbookbox	7.1081	2.8031
3	boardgame	11.5233	6.7706
4	book	2.3583	0.0000
5	breadcartoychips	14.0717	5.5274
6	breadcube	2.8802	0.1116
7	breadcubechips	2.9957	2.4348
8	breadtoy	4.9757	0.7951
9	breadtoycar	6.5060	2.0060
10	carchipscube	3.8545	0.2788
11	cube	5.9570	1.4139
12	cubebreadtoychips	9.8563	3.0795
13	cubechips	7.5246	1.3697
14	cubetoy	8.5944	2.2892
15	dinobooks	19.0722	14.5556
16	game	8.8884	1.3734
17	gamebiscuit	4.9970	2.3018
18	toycubecar	12.0600	1.3900
	Mean	7.3584	2.5535
	Median	6.5060	1.4139

Table 6.1: T-Linkage Misclassification Error % for two-view motion segmentation. These results are the average of 100 iterations for each sampling strategy. The lower the better.

6.2.2 Comparison with Multi-GS

In order to make a fair comparison with Multi-GS, we set the same number of sampled MSSs for the two methods. Specifically, we set them at the double of the number of points. For each experiment we counted the number of pure MSS sampled in the process, as shown in Eq. 6.1. In Tab. 6.2 we reported the overall number of pure MSS for Multi-GS, localized and semantic-aware sampling. We aim to maximize the number of pure MSS, so the higher is the result, the better is the method. Looking at the table we see that, by using the same number of samplings, our method is able to retrieve a larger number of pure MSS than Multi-GS.

Index	**Label**	**Multi-GS**	**Localized**	**Semantic-aware**
0	biscuit	138	36	461
1	biscuitbook	199	24	260
2	biscuitbookbox	179	35	288
3	boardgame	138	34	229
4	book	122	26	133
5	breadcartoychips	84	14	165
6	breadcube	196	64	284
7	breadcubechips	118	21	191
8	breadtoy	205	57	432
9	breadtoycar	75	10	160
10	carchipscube	107	49	157
11	cube	82	2	32
12	cubebreadtoychips	256	34	287
13	cubechips	172	41	346
14	cubetoy	169	62	321
15	dinobooks	92	24	352
16	game	33	0	4
17	gamebiscuit	184	12	195
18	toycubecar	107	46	194

Table 6.2: Comparison Multi-GS, Localized and Semantic-aware Sampling for the motion segmentation task using data from [25]. We report in the table the overall number of pure MSS (all the points in MSS belong to the same model) for all the strategies. The higher the better.

6.2.3 Comparison with DGSAC

We compared DGSAC with our implementation of T-Linkage with semantic-aware sampling. For a fair comparison we directly used the original code of DGSAC that we have downloaded from [51]. A complete overview of the results is shown in Tab. 6.3. We run DGSAC 100 times for any image pair of [25] for the motion segmentation task. In order to avoid fluctuations due to particularly lucky or unlucky runs, the results are the average of 100 iterations. At the bottom of the table, we reported also the overall mean and median. Since we aim to minimize the misclassification error, the results in the table show that T-Linkage equipped with the semantic-aware sampling surpassed DGSAC.

Index	Label	DGSAC	Semantic-aware
0	biscuit	2.6545	0.0152
1	biscuitbook	2.0205	0.0000
2	biscuitbookbox	1.8764	2.8031
3	boardgame	19.0573	6.7706
4	book	1.3850	0.0000
5	breadcartoychips	11.3165	5.5274
6	breadcube	2.3926	0.1116
7	breadcubechips	6.6043	2.4348
8	breadtoy	6.3229	0.7951
9	breadtoycar	10.8916	2.0060
10	carchipscube	14.6303	0.2788
11	cube	6.6854	1.4139
12	cubebreadtoychips	9.4832	3.0795
13	cubechips	3.4472	1.3697
14	cubetoy	2.8273	2.2892
15	dinobooks	17.4861	14.5556
16	game	5.5408	1.3734
17	gamebiscuit	3.6707	2.3018
18	toycubecar	9.6750	1.3900
	Mean	7.2615	2.5535
	Median	6.3229	1.4139

Table 6.3: T-Linkage Misclassification Error % for two-view motion segmentation. These results are the average of 100 iterations for each sampling strategy. The lower the better.

6.2.4 Comparison with CONSAC

We compared CONSAC with our implementation of T-Linkage with localized and semantic-aware sampling. Unfortunately, the implementation of CONSAC (taken from [44]) do not implement the motion segmentation task. Thus, to keep a clean comparison between the methods, we applied our method to the plane segmentation task. A complete overview of the results is shown in Tab. 6.4. The table show that T-Linkage equipped with the semantic-aware sampling and CONSAC have comparable results for the plane segmentation. The rationale is that the network is not trained to recognize planes, thus we can only distinguish between foreground (buildings) and background (sky or grass).

Index	**Label**	**CONSAC**		**Localized**		**Semantic-aware**	
		avg.	std.	avg.	std.	avg.	std.
0	barrsmith	2.74	1.16	8.96	5.35	1.58	0.55
2	bonython	0.00	0.00	2.22	1.65	0.00	0.00
3	elderhalla	11.31	8.47	17.57	8.78	0.75	0.23
4	elderhallb	0.94	0.44	16.78	4.72	16.94	6.85
5	hartley	11.69	0.16	4.56	2.54	1.69	0.85
6	johnsona	17.64	0.11	7.83	2.16	6.92	3.85
7	johnsonb	20.22	6.24	17.38	3.35	14.73	3.02
8	ladysymon	2.95	0.00	9.45	0.95	10.72	1.21
9	library	0.93	0.00	5.49	1.67	5.40	0.96
10	napiera	7.02	4.33	8.41	1.20	6.36	1.41
11	napierb	10.97	2.00	17.37	2.28	11.89	1.87
12	neem	4.56	3.00	16.27	6.75	6.56	4.13
13	nese	0.00	0.00	0.79	1.39	2.28	1.20
14	oldclassicswing	1.32	0.00	4.91	1.06	0.26	0.00
15	physics	0.00	0.00	0.57	0.46	0.00	0.00
16	sene	1.60	2.00	4.08	7.17	0.72	0.30
18	unionhouse	0.30	0.00	3.55	3.66	0.30	0.00
		5.54	1.60	8.60	3.24	5.12	1.55

Table 6.4: Comparison CONSAC and T-Linkage with Localized and Semantic-aware Sampling.

Chapter 7

Conclusions and future works

In this work, we have presented a novel sampling strategy for robust fitting, which we call *semantic-aware sampling.* We explicitly assume that the input points have been extracted from a picture (set of pictures). This limits the applicability of our method but allows to reduce the number of required samplings and, when embedded into robust estimators, to achieve a lower misclassification error. In addition, our method allows us to increase the number of pure minimal sample set (MSS), namely set of points all belonging to the same geometric model (neither gross-outliers nor pseudo-outliers).

We propose to combine the hand-crafted approach where a set of corresponding points is extracted from the pictures and the data-driven approach where we leverage the feature maps of a *Convolutional Neural Network* to get a set of probability maps highlighting the input regions that are likely to contain an object (or multiple objects) and are therefore worth sampling. We can inspect features at different levels of abstraction and aggregation, therefore, for the sake of generality, we call this information *semantics*, referring to the semantics of the points as opposed to their geometric configuration.

The results of our experiments showed that, fixed the number of samplings, the use of image semantics is effective in producing many more pure MSS and reducing misclassification error. We have taken T-Linkage as our reference algorithm and we would like to investigate what the effect of this sampling strategy would be in other state-of-the-art methods, both consensus-based and preference-based. Furthermore, it would be interesting to apply our technique to the video segmentation task, taking into account a set of contiguous frames and segmenting the rigid motion of objects over time.

From the Deep Learning side, we believe that being able to produce more accurate semantics can further improve the performance of the sampling. There are several directions we can take: producing higher resolution maps [35][41], or following other approaches instead of using features or CAMs, such as using autoencoders [5] that learn a compressed representation of the input, thus learning hidden structures in the data. This approach would allow us to be even more general and we believe it would be an interesting research direction.

Bibliography

[1] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[2] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.

[3] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[4] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T.

[5] M. Kramer. “Nonlinear principal component analysis using autoassociative neural networks”. In: *Aiche Journal* 37 (1991), pp. 233–243.

[6] M. Leshno et al. “Multilayer Feedforward Networks with a Non-Polynomial Activation Function Can Approximate Any Function”. In: *Neural Networks.* 1993.

[7] Tony Lindeberg. “Scale-Space Theory: A Basic Tool for Analysing Structures at Different Scales”. In: *Journal of Applied Statistics* 21 (Sept. 1994), pp. 224–270. DOI: 10.1080/757582976.

[8] Tom M. Mitchell. *Machine Learning.* New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[9] David G. Lowe. “Object Recognition from Local Scale-Invariant Features”. In: *Proc. of the International Conference on Computer Vision ICCV, Corfu.* 1999.

[10] Allan Pinkus. “Approximation theory of the MLP model in neural networks”. In: vol. 8. Cambridge University Press, 1999, pp. 143–195. DOI: 10.1017/S0962492900002919.

[11] Andrew Fitzgibbon and Andrew Zisserman. “Multibody Structure and Motion: 3-D Reconstruction of Independently Moving Objects”. In: May 2000. ISBN: 978-3-540-67685-0. DOI: 10.1007/3-540-45054-8_58.

[12] Ondrej Chum, Jiri Matas, and Josef Kittler. “Locally optimized RANSAC”. In: vol. 2781. Sept. 2003, pp. 236–243. ISBN: 978-3-540-40861-1. DOI: 10.1007/978-3-540-45243-0_31.

[13] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. USA: Cambridge University Press, 2003. ISBN: 0521540518.

[14] Li Fei-Fei, R. Fergus, and P. Perona. “Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories”. In: *CVPR Workshops*. 2004.

[15] Yasushi Kanazawa and Hiroshi Kawakami. “Detection of Planar Regions with Uncalibrated Stereo using Distributions of Feature Points”. In: (Jan. 2004). DOI: 10.5244/C.18.27.

[16] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[17] Marco Zuliani, C.S. Kenney, and B. Manjunath. “The multiRANSAC algorithm and its application to detect planar homographies”. In: Oct. 2005, pp. III–153. ISBN: 0-7803-9134-9. DOI: 10.1109/ICIP.2005.1530351.

[18] G. Griffin, A. Holub, and P. Perona. “Caltech-256 Object Category Dataset”. In: 2007.

[19] Roberto Toldo and Andrea Fusiello. “Robust Multiple Structures Estimation with J-Linkage”. In: *Computer Vision – ECCV 2008*. Ed. by David Forsyth, Philip Torr, and Andrew Zisserman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 537–547. ISBN: 978-3-540-88682-2.

[20] J. Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[21] Marius Muja and David Lowe. “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.” In: vol. 1. Jan. 2009, pp. 331–340.

[22] Emilio Soria Olivas et al. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes.* Information Science Reference - Imprint of: IGI Publishing, 2009. ISBN: 1605667668.

[23] Tat-Jun Chin, Jin Yu, and David Suter. “Accelerated Hypothesis Generation for Multi-structure Robust Fitting”. In: *Computer Vision – ECCV 2010.* Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 533–546. ISBN: 978-3-642-15555-0.

[24] K. E. Ozden, K. Schindler, and L. Van Gool. “Multibody Structure-from-Motion in Practice”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.6 (2010), pp. 1134–1141. DOI: 10.1109/TPAMI.2010.23.

[25] H. S. Wong et al. “Dynamic and hierarchical multi-structure geometric model fitting”. In: *2011 International Conference on Computer Vision.* 2011, pp. 1044–1051. DOI: 10.1109/ICCV.2011.6126350.

[26] R. Raguram et al. “USAC: A Universal Framework for Random Sample Consensus”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 2022–2038. DOI: 10.1109/TPAMI.2012.257.

[27] L. Magri and A. Fusiello. “T-Linkage: A Continuous Relaxation of J-Linkage for Multi-model Fitting”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* 2014, pp. 3954–3961. DOI: 10.1109/CVPR.2014.505.

[28] Christian Szegedy et al. *Going Deeper with Convolutions.* 2014. arXiv: 1409.4842 [cs.CV].

[29] Zhe Xu et al. “Architectural Style Classification Using Multinomial Latent Logistic Regression”. In: *Computer Vision – ECCV 2014.* Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 600–615.

[30] Yann LeCun, Y. Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[31] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision.* 2015. arXiv: 1512.00567 [cs.CV].

[32] Bolei Zhou et al. "Learning Deep Features for Discriminative Localization". In: *CoRR* abs/1512.04150 (2015). URL: `http://arxiv.org/abs/1512.04150`.

[33] Martín Abadi et al. *TensorFlow: A system for large-scale machine learning*. 2016. arXiv: `1605.08695 [cs.DC]`.

[34] Rudolf Kruse et al. *Computational Intelligence: A Methodological Introduction*. 2nd. Springer Publishing Company, Incorporated, 2016. ISBN: 1447172949.

[35] Ramprasaath R. Selvaraju et al. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391 (2016). arXiv: `1610.02391`. URL: `http://arxiv.org/abs/1610.02391`.

[36] Amogh Gudi et al. "Object-Extent Pooling for Weakly Supervised Single-Shot Localization". In: *CoRR* abs/1707.06180 (2017). URL: `http://arxiv.org/abs/1707.06180`.

[37] Zhou Lu et al. *The Expressive Power of Neural Networks: A View from the Width*. 2017. arXiv: `1709.02540 [cs.LG]`.

[38] Suo Qiu. "Global Weighted Average Pooling Bridges Pixel-level Localization and Image-level Classification". In: *ArXiv* abs/1809.08264 (2018). URL: `http://arxiv.org/abs/1809.08264`.

[39] Lokender Tiwari and S. Anand. "DGSAC: Density Guided Sampling and Consensus". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018), pp. 974–982.

[40] Florian Kluger et al. "CONSAC: Robust Multi-Model Fitting by Conditional Sample Consensus". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[41] Pietro Morbidelli et al. "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), pp. 4067–4071.

[42] Anastasis Kratsios. "The Universal Approximation Property: Characterization, Construction, Representation, and Existence". In: *Annals of Mathematics and Artificial Intelligence* (Jan. 2021). DOI: `10.1007/s10472-020-09723-1`.

[43] Vinny DaSilva. *Computer Vision for Busy Developers*. `https://medium.com/@vad710/cv-for-busy-devs-scale-space-9368e3be938b`. Accessed: 2020-12-6.

[44] Florian Kluger et al. *CONSAC*. `https://github.com/fkluger/consac`. Accessed: 2020-03-12.

[45] Matplotlib. *Matplotlib: Visualization with Python*. `https://matplotlib.org/stable/index.html`. Accessed: 2020-03-12.

[46] Ashwin Naidu. *Image Filtering*. `https://github.com/ashushekar/image-convolution-from-scratch`. Accessed: 2020-02-6.

[47] Numpy. *What is NumPy?* `https://numpy.org/doc/stable/user/whatisnumpy.html`. Accessed: 2020-03-12.

[48] OpenCV. *OpenCV: About*. `https://opencv.org/about/`. Accessed: 2020-03-12.

[49] Utkarsh Sinha. *SIFT: Theory and Practice*. `https://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/`. Accessed: 2020-12-6.

[50] TensorFlow. *Transfer learning and fine-tuning*. `https://www.tensorflow.org/tutorials/images/transfer_learning`. Accessed: 2020-03-12.

[51] Lokender Tiwari and S. Anand. *Implementation of DGSAC: Density Guided SAmpling and Consensus, IEEE WACV 2018*. `https://bitbucket.org/lokender/dgsac`. Accessed: 2020-03-12.

[52] Shiva Verma. *Understanding 1D and 3D Convolution Neural Network | Keras*. `https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610`. Accessed: 2020-02-6.

[53] Stanford Vision and Learning Lab. *CS231n Convolutional Neural Networks for Visual Recognition*. `https://cs231n.github.io/convolutional-networks/`. Accessed: 2020-02-6.

[54] Wikipedia. *pandas (software)*. `https://en.wikipedia.org/wiki/Pandas_(software)`. Accessed: 2020-03-12.