Master in Computer Vision Barcelona

# Week 1. Bag Of Visual Words

C3. Machine Learning for Computer Vision

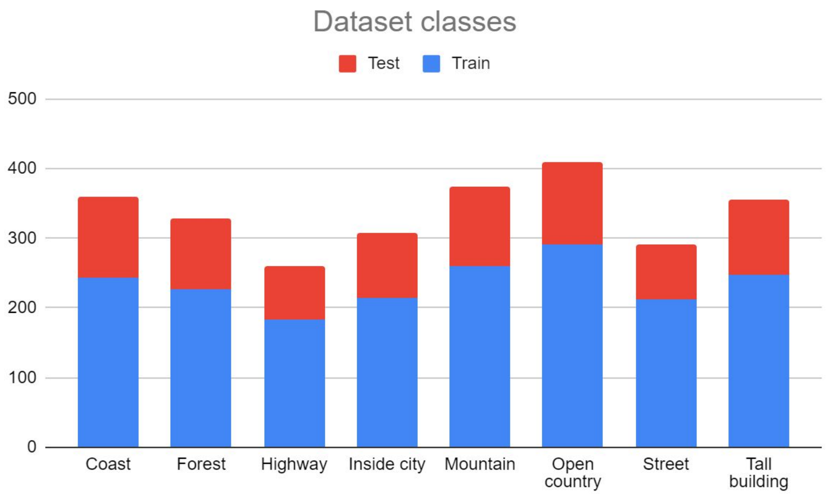

**Group 2**

Cristina Aguilera
Jordi Morales
Ainoa Contreras

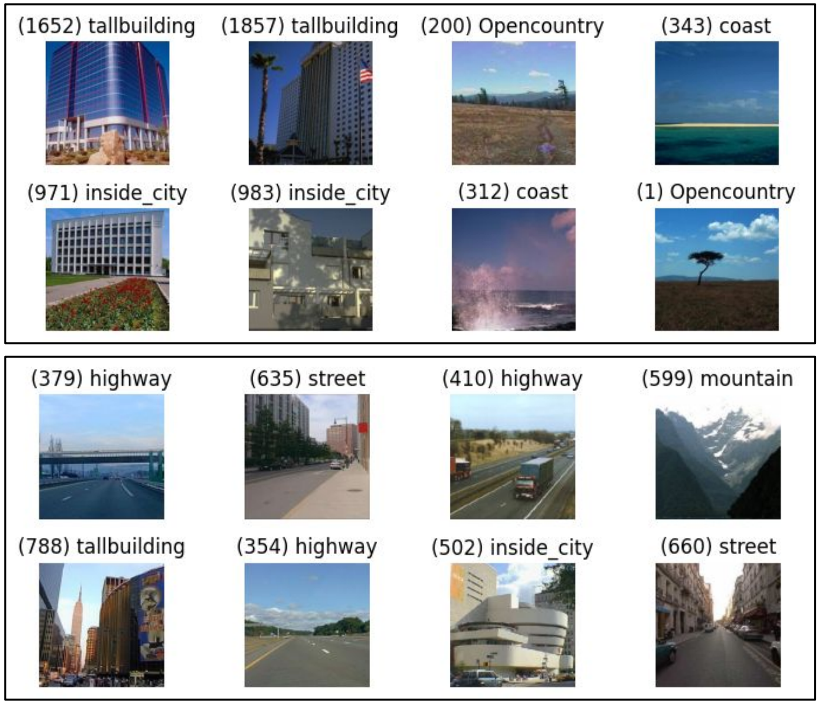
# Table of Contents

# Dataset

The dataset is considerably **balanced**. Checked the distribution of classes across train and test dataset.




Random visualization of sets: clearly detect visual part of the images that are **similar across same class** and **different in between classes**.



# Model Pipeline and Hyperparameters

Huge number of hyperparameters stored in **config dictionary**

```
config = {
        'descriptor': 'dense_SIFT',
        'descriptor_param': 771,
        'k_codebook': 348,
        'step_size': 10,
        'scale_factor': 8,
        'distance': 'jaccard',
        'n_folds': 4,
        'normalize': None,
        'scale': None,
        'dim_reduction': None,
        'n_components': None,
        'model_type': "svm",
        'C': C,
        'gamma': gamma,
        'kernel': "rbf",
        'pyramid_level': 0
    }
```

**Spatial Pyramids** → **Compute Descriptors** → **Create Visual Words** → **Dimensionality Reduction** → **Classifier** → **Cross Validation**

**Spatial Pyramids**
- Pyramid Level

**Compute Descriptors**
- Type of descriptor
- Threshold
- Number of features
- Step Size
- Scale Factor

**Create Visual Words**
- Codebook size k
- Normalizer
- Scaler

**Cross Validation**
- Number of CV fold

**Classifier**
- Model Type
- Number of k-neighbors
- Distance metric
- SVM kernel
- Models parameters: Alpha, gamma, C…

**Dimensionality Reduction**
- Dimensionality reduction algorithm
- Number of components

# Hyperparameters Optimization - Experimental Approaches

Two different types of experiments:

1. **Individual influence of hyperparameters**: Test different amounts for each hyperparameter and check its effect into the general model. Fix the best value and use it for the rest of experiments.

2. Try to **optimize all hyperparameters at once** to find the best possible model.

### Grid Search

Categorical/Boolean hyperparameters or numerical with limited value range:

- Type of Descriptor (SIFT, AKAZE, denseSIFT…)
- Distances for KNN model
- Use of normalizer and/or scaler
- …

### Random Search

Numerical hyperparameters with large value ranges:

- Step Size and Scale Factor
- Codebook size k
- Number of k neighbors for KNN
- …

# Hyperparameters Optimization - Frameworks

Generic scheme for hyperparameters search:

```python
# Define the objective function (classification accuracy)
def objective(trial):
    step_size = trial.suggest_int('step_size', 1, 100)
    scale_factor = trial.suggest_int('scale_factor', 2, 20)

    config = {
            'descriptor': 'dense_SIFT', 'descriptor_param': 771, 'k_codebook': 128,
            'step_size': step_size, 'scale_factor': scale_factor, 'k_neigh': 5,
            'distance': 'euclidean', 'n_folds': 10, 'normalize': None, 'scale': None,
            'dim_reduction': None, 'model_type': "knn", 'pyramid_level': 0
        }

    accuracy, precision, recall, f1_score = cross_validation(train_images_filenames, train_labels, config)

    # Log hyperparameters and metrics to Weights & Biases
    wandb.log({'step_size': step_size, 'scale_factor':scale_factor, 'f1-score': f1_score,
               'accuracy': accuracy, 'precision': precision, 'recall': recall})

    return f1_score

# Initialize Weights & Biases
wandb.init(project='c3_2', name='dense_SIFT step_size and scale_factor optimization',
           settings=wandb.Settings(start_method="fork"))

# Create an Optuna study
study = optuna.create_study(direction='maximize')

# Optimize the objective function
study.optimize(objective, n_trials=50)

# Access the best trial and its parameters
best_trial = study.best_trial
```

OPTUNA

Suggest hyperparameters values using trial object

Create a study object and invoke the optimize method over 50 trials

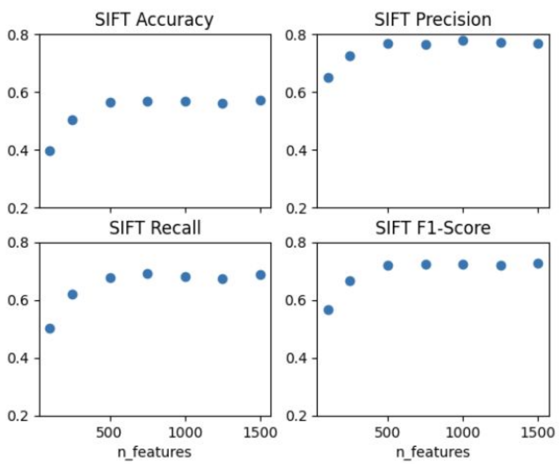
W&B

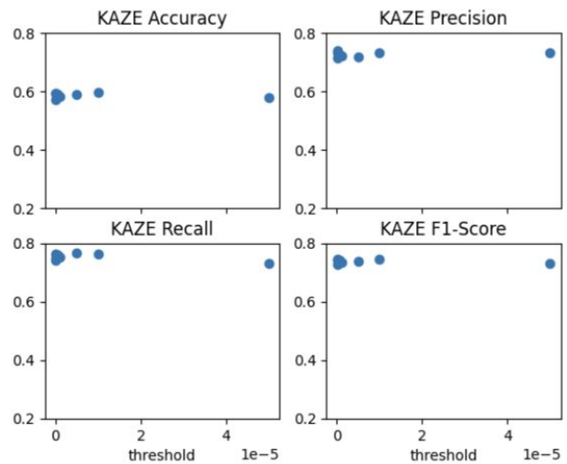
Log metrics over time to visualize performance

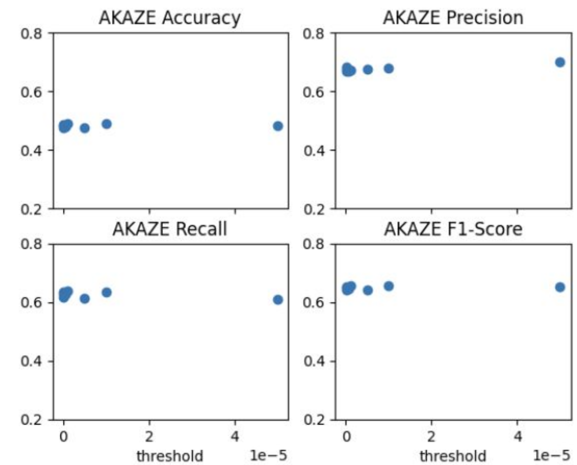
Start a w&b run

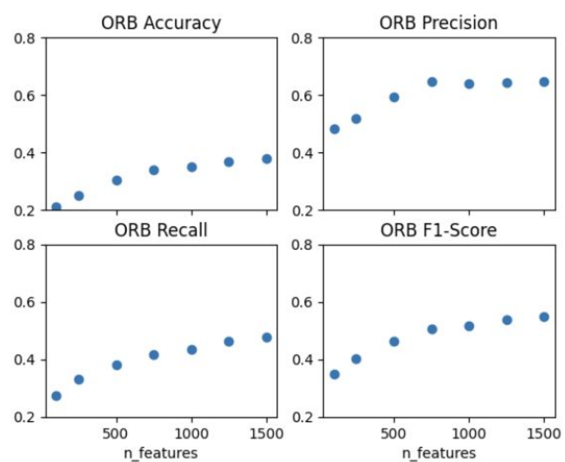
# Choosing the best descriptor (I)

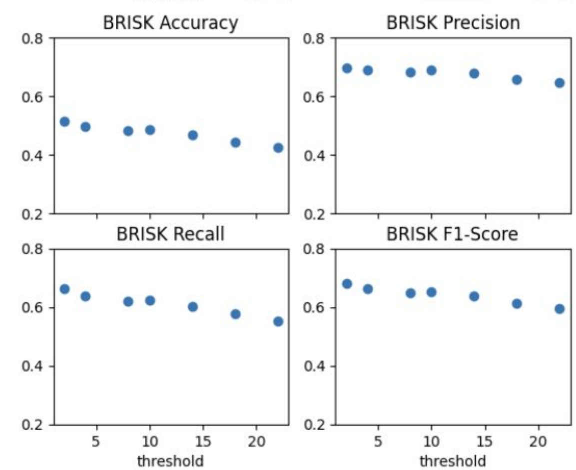
We have evaluated the following descriptors depending on
- number of features
- threshold




The number of features that one descriptor computes for each image is determined in some cases, by the parameter **n_features**, and in some other cases by the parameter **threshold** (this value determines the limit of one feature to be considered).

# Choosing the best descriptor (II)

In the plots shown in the last slide we can observe that as we **increase the number of features** or **decrease the threshold** we obtain better performance. This is due to the fact that the greater the number of features, the more specific the model is.

| RANKING | DESCRIPTOR | F1-SCORE |
|---|---|---|
| 1 | KAZE | 74.6% |
| 2 | SIFT | 72.6% |
| 3 | BRISK | 68% |
| 4 | AKAZE | 65,6% |
| 5 | ORB | 54,8% |

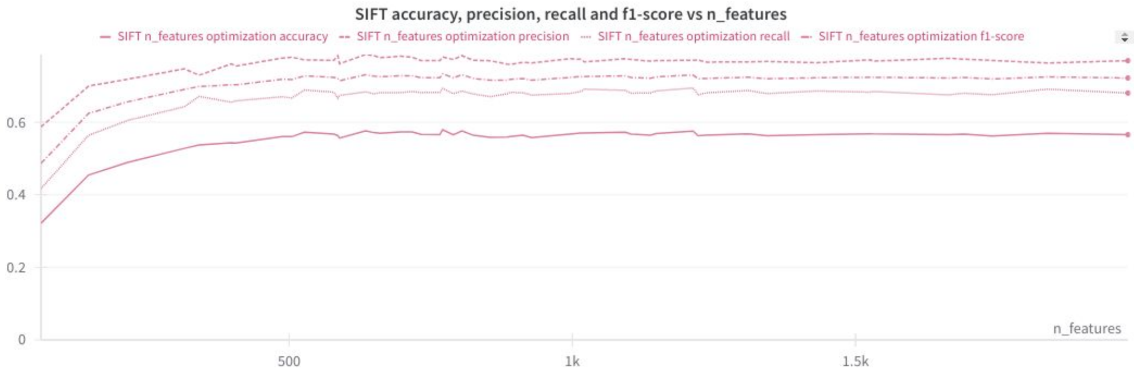
We will focus on the first two: **KAZE** and **SIFT**

We have performed the accuracy, precision, recall and F1-score metrics for evaluation, in this task we will compare the models using **F1-score** since it is the measure of the predictive performance that encompasses precision and recall, that is, that gives more information in one single metric.

# Choosing the best descriptor (III)

SIFT

```
n_features = trial.suggest_int('n_features', 50, 2000)
```



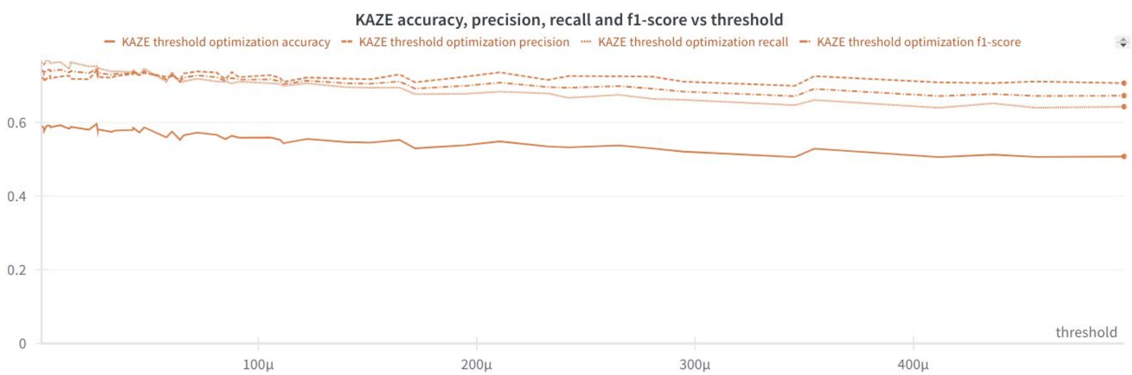

Optimizing search using OPTUNA

BEST F1-score = **73.35%**

**771** features

KAZE

```
threshold = trial.suggest_float('threshold', 0.0000005, 0.0005)
```




BEST F1-score = **74.65%**

threshold of **0.000025943**

KAZE outperforms SIFT regarding the F1-score, but it is computationally more expensive. Since the there is <1% of difference between values, **SIFT descriptor is preferred**.

# Dense SIFT (I)

- SIFT → extracts main features automatically from the image
- Dense SIFT → extracts features given a grid defined by the step_size and the scale_factor
    - step_size: spacing between the centers of neighboring keypoint regions in the image grid
    - scale_factor: size of the keypoint regions at different scales

| | SIFT | dense_SIFT |
|---|---|---|
| Accuracy | 57.94% | 74.96% |
| Precision | 77.98% | 85.24% |
| Recall | 69.31% | 86.23% |
| F1-score | 73.35% | 85.67% |

Random values:
step_size = 5
scale_factor = 10

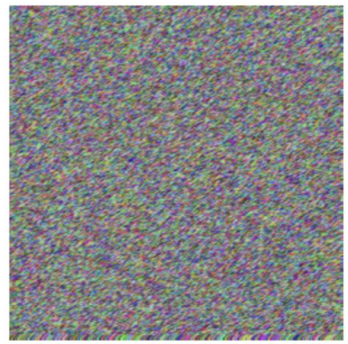
Dense_SIFT outperforms SIFT

# Dense SIFT (II)

The main benefits of computing descriptors using one grid is the **spatial information** and the **quantity of shapes and colors**. In the example, we could determine if the image refers to a beach if we focus on two big groups of descriptors: sky and sea.

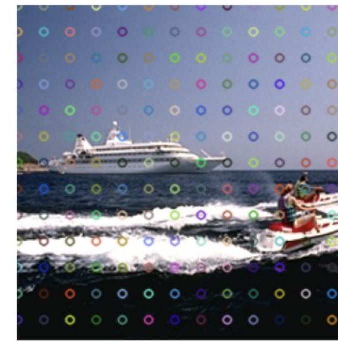
original image

SIFT



Dense SIFT
step_size = 1

Dense SIFT
step_size = 10



Dense SIFT
step_size = 20

**TRADE-OFF**: the smaller the step, the more information we get from the image, and we could think that a step size of 1 might be the best option. This is not a good idea because we are focusing too much on small details and, furthermore, the computational cost increases.

Optimizing search using OPTUNA

```
step_size = trial.suggest_int('step_size', 1, 100)
scale_factor = trial.suggest_int('scale_factor', 2, 20)
```

BEST F1-score = **85.54%**
step_size = **10**
scale_factor = **8**

# Normalizer and Scaler

| Normalization | Scalation |
|---|---|
| Scales each sample to have a unit norm (L2) | Transforms data to have a mean of zero and a standard deviation of one |

Invariant to changes in magnitude
Comparable and suitable features
Useful for algorithms that are sensitive to the scale of features

| | Dense_SIFT | Normalized dense_SIFT | Scaled dense_SIFT |
|---|---|---|---|
| F1-score | 85.54% | 84.61% | 86.97% |

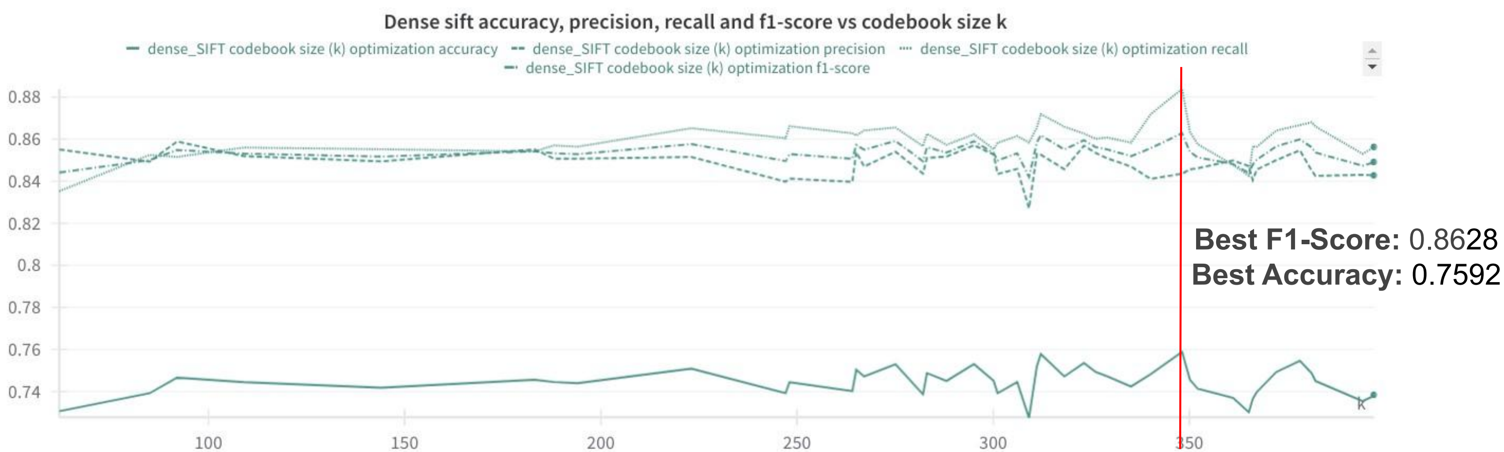
**Normalization** and **scalation** of the visual words **DOES NOT IMPROVE** considerably the model predictions

some reasons

- inherently insensitive algorithm
- features already within a similar range naturally
- improvement gained from normalization or scaling might be overshadowed by other hyperparameters or aspects of the model that need tuning
- normalization or scaling might not be sufficient (non-Gaussian data or it has outliers)
- inappropriate scaling/normalization method

# Visual Words: Codebook size k

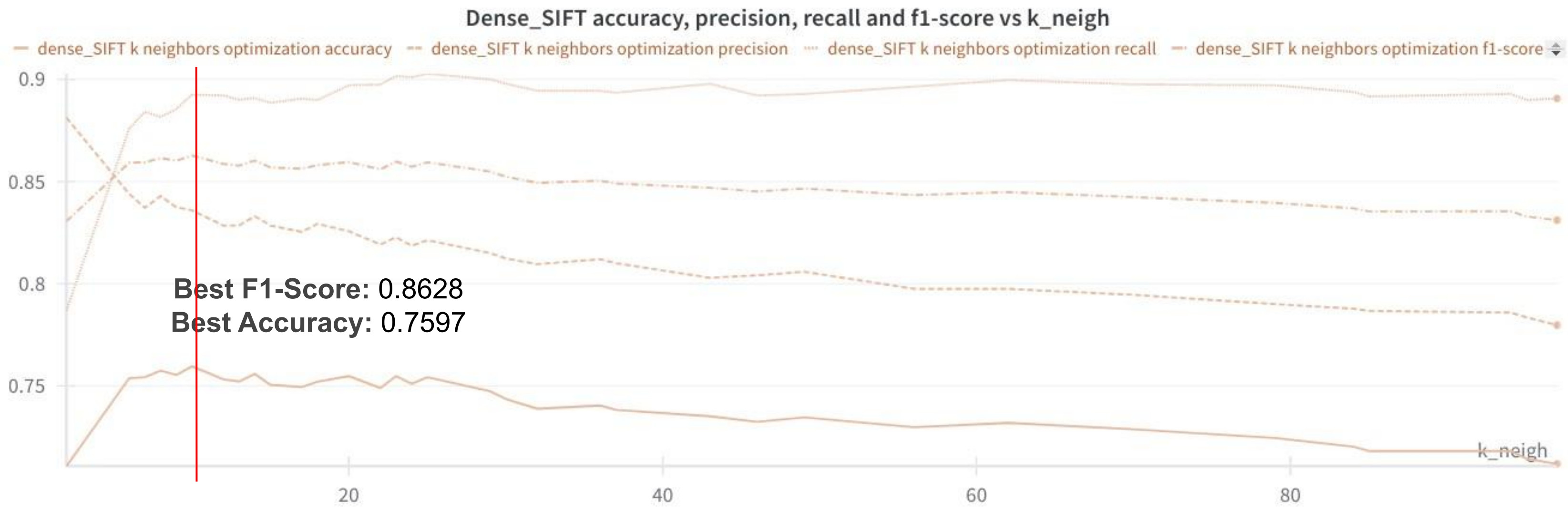
- Small codebook size → Visual words are too general, not representative enough to perform classification properly (flat curves)
- Large codebook size → Visual words are too specific, so the features are splitted into irrelevant details which leads to decrease in the performance (abrupt changes in the curves).




**Best results** → Codebook size k = 348

# Classifier: KNN - Number of k-neighbors

We observe a small increase until we reach k=10 and then they start declining as we increase k (except for the recall). We should choose it to be large enough so that the noise in the data is minimized, but small enough so that samples from other classes are not included.

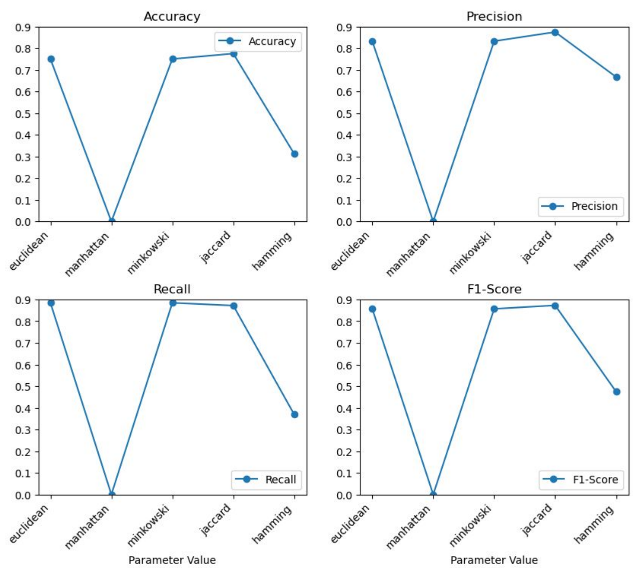



**Best results** → Number of k-neighbors = 10

# Classifier: KNN - Distance metric

- Most of the distances worked in a similar way, except for Manhattan and Hamming.
- We obtain the best results with Jaccard distance by a small margin. We have been able to improve the f1-score by slightly less than 2%.
- Remark also that Minkowski and Euclidean obtained the same results since the default configuration of parameters of the first make it perform as the second.

| | Euclidean | Jaccard |
|---|---|---|
| Accuracy | 75% | 77.5% |
| Precision | 83.25% | 87.45% |
| Recall | 88.47% | 87.2% |
| F1-score | 85.68% | 87.3% |

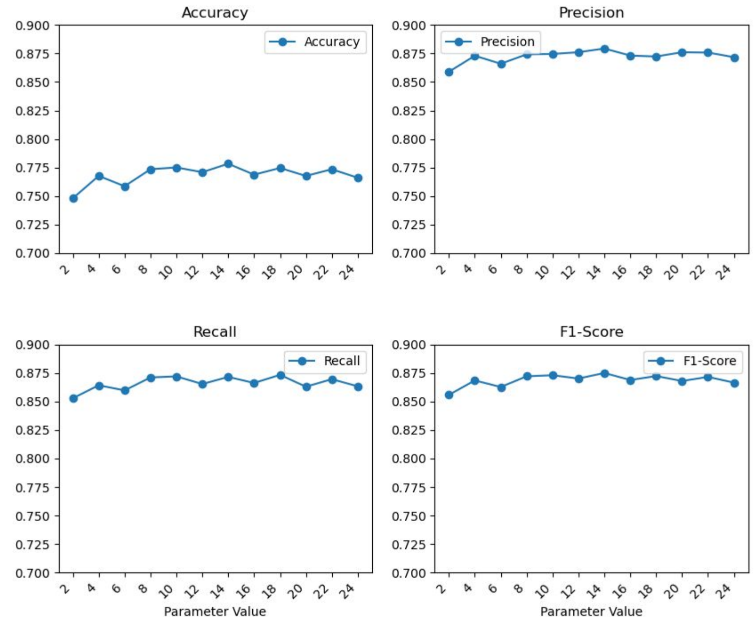



**Best results** → Distance metric = jaccard

# Cross Validation

Until now, we have been using CV with 10 folds to optimize the different hyperparameters of the model. As it is a critical step, we would like to check the influence of the number folds used. For this experiment, we have used the best combination of parameters until the moment.

- Most of the values performed really similar, with less than a 2,5% difference between all of them.
- As the number of folds increase, also does the computational time. For that reason we decided to use a small number of folds.



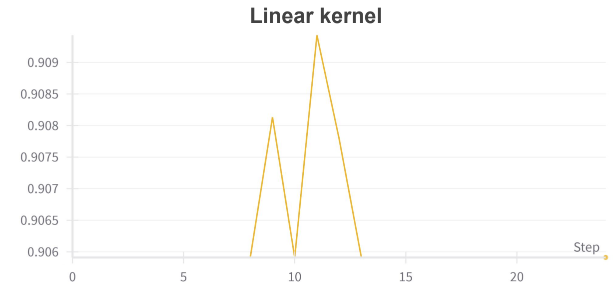

**Best results** → CV folds = 4

# Other Models: SVM

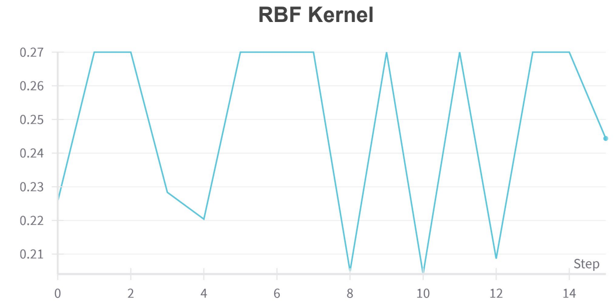
We explore different kernels and configurations of the SVM Classifier model. We use the best parameters found in previous experiments and use Optuna for optimizing the new hyperparameters:

- The Histogram intersection kernel yields the best performance, although it is quite sensitive to changes in its hyperparameters.
- The Linear kernel model obtains very similar results regardless of hyperparameter chosen.
- The RBD kernel model is the worst by far, obtaining the worse results seen in any test so far.

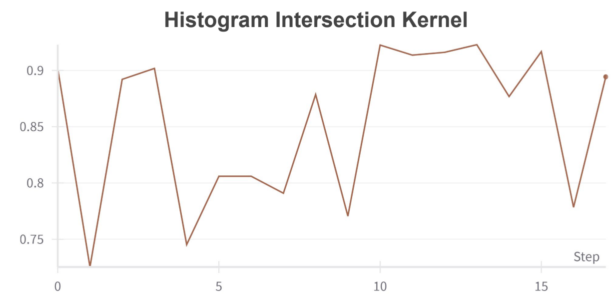
**Best results** → Histogram Intersection, C = 0.13146, *alpha* = 0.7




**Best F1-Score:** 0.9059
**Best Accuracy:** 0.8341




**Best F1-Score:** 0.2700
**Best Accuracy:** 0.1563




**Best F1-Score: 0.9228**
**Best Accuracy: 0.8570**

# Other Models: Logistic Regression

We also tried to fit a Logistic Regression model:

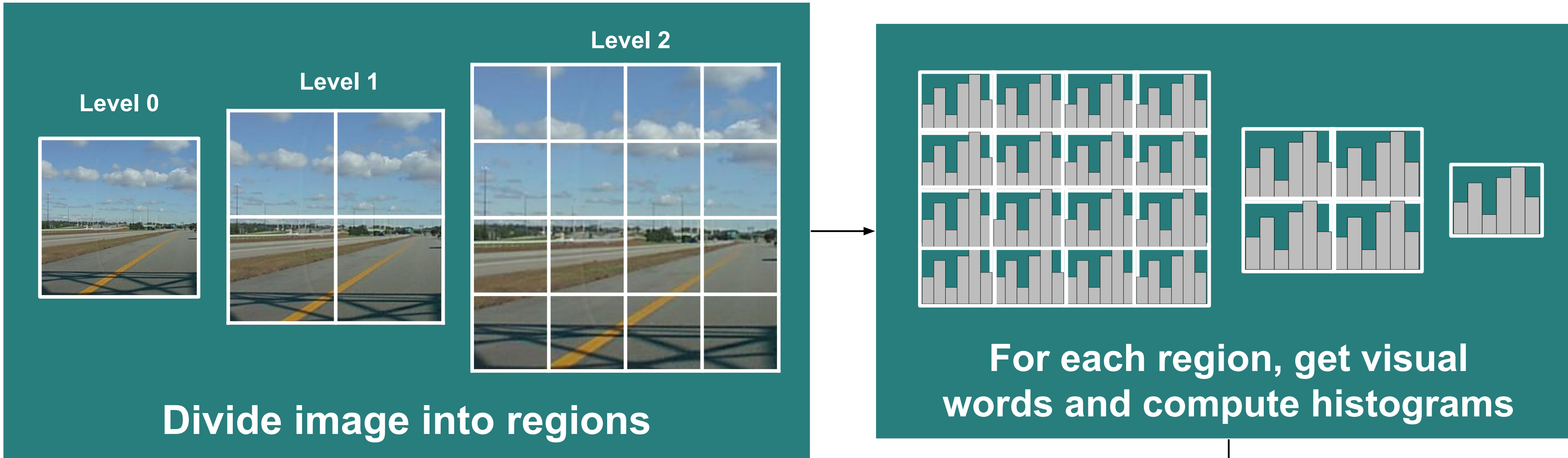
- We find the best results with a regularization parameter of 599.21, without including an intercept and using a Newton method for training.

- Despite the diversity of hyperparameters to choose, the model obtains a very similar score in any case.

- Still, this model performs slightly worse than the SVM with a Histogram Intersection kernel.

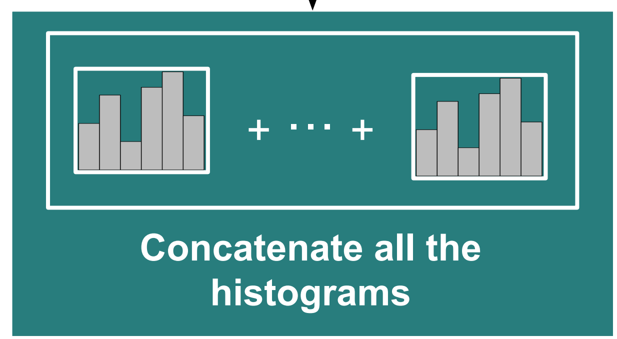

**Best F1-Score: 0.9176**
**Best Accuracy: 0.848**

# Spatial Pyramids: Approach




The idea of a **spatial pyramid** is to extract features at different scales to **add structural and spatial information** to the final descriptor:

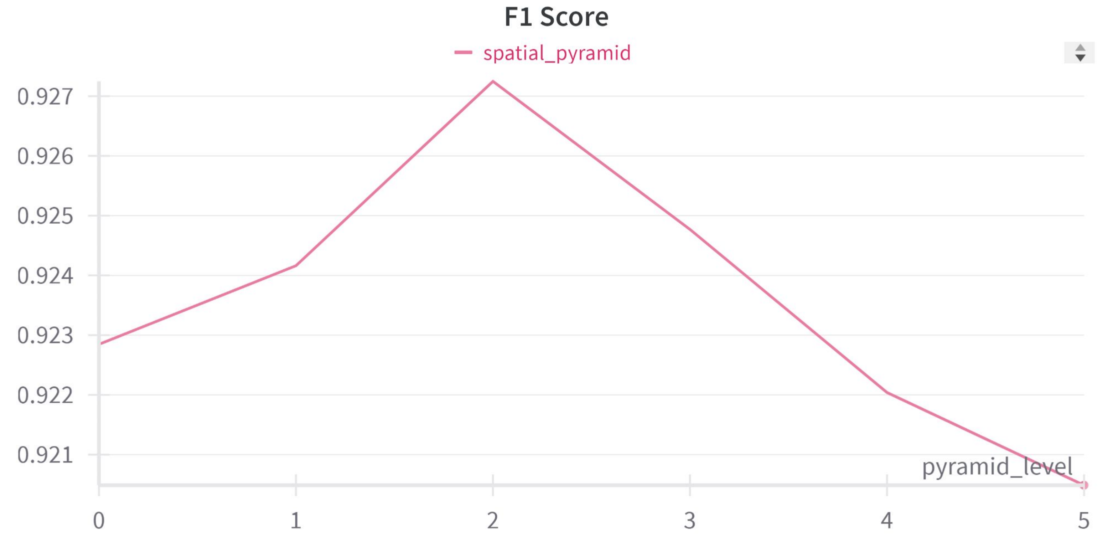
- The visual words detected in each region come from a global bag of visual words created using all the whole images.
- The algorithm is defined by a *level* parameter, which determines the “height” of the pyramid.



# Spatial Pyramids: Results

We obtain the best results, F1-Score of  0.9272, when using a pyramid level of 2, an improvement with respect to not using a spatial pyramid, although it is a very slim difference.

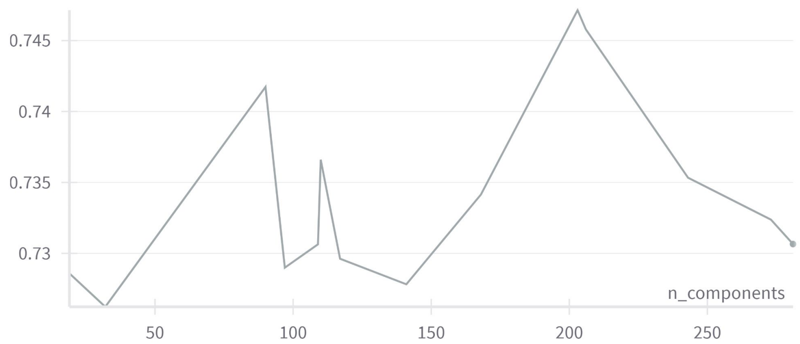
With larger values, the model gets worse and worse, both in the results that produces (Curse of dimensionality, sparse representation) and the execution time.




**Best results** → Pyramid level = 2

# Dimensionality Reduction: LDA and PCA

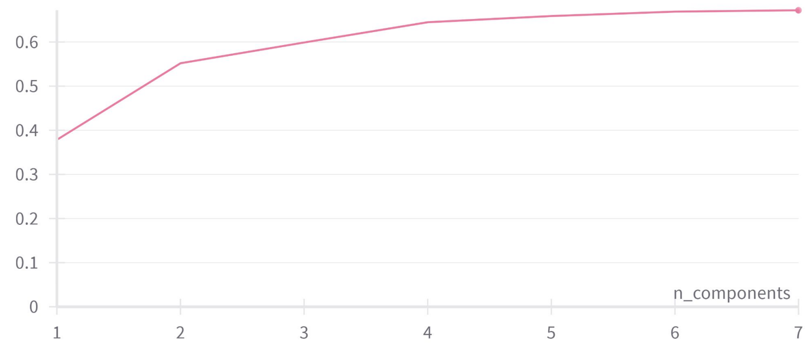
## PCA




**Best F1-Score:** 0.7471
**Best Accuracy:** 0.5965

- **Number of components explored:** Between 1 and 300.
- We haven't found a **number of components** in the range that we have explored that could yield better results than the model that we already had.

## LDA




**Best F1-Score:** 0.6720
**Best Accuracy:** 0.5061

- **Number of components explored:** 1 to 7.
- The LDA algorithm is **not able to provide a good representation of the image features**, regardless of the number of components. The performance plateaus with the largest amount of components (7).

# Fisher Vectors

| | Bag of Visual Words | Fisher Vectors |
|---|---|---|
| Descriptors modelization | **Notebook**: clusterization of the extracted descriptors using K-means | **Gaussian Mixture Model**: probabilistic model trained with the extracted descriptors |
| Visual representation | **Histogram**: frequency of occurrence of visual words | **Fisher Vector**: first-order statistics (mean deviations), and second-order statistics (variance deviations) |
| Performance and computational cost | Worst performance, low computational cost | Better performance, more computational cost |

We have not obtained evaluation metrics from our experiments, therefore **we cannot** quantitatively **compare** both techniques.

# Test Evaluation - Quantitative Results

Once we have all our hyperparameters optimized, we need to truly evaluate the model with the test dataset

- Train the model using all the training data and evaluate it on the test set.
- We obtain a **F1-Score of 0.9318**, almost 0.5% higher than what we found during cross-validation. This is good news, indicating that our model has good generalization capabilities and is able to work well with unseen data.

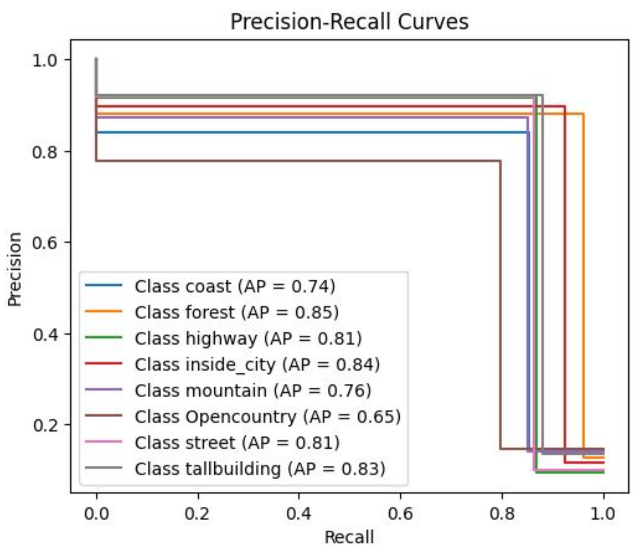
**After all the experiments, we have found the next configuration to yield the better results:**

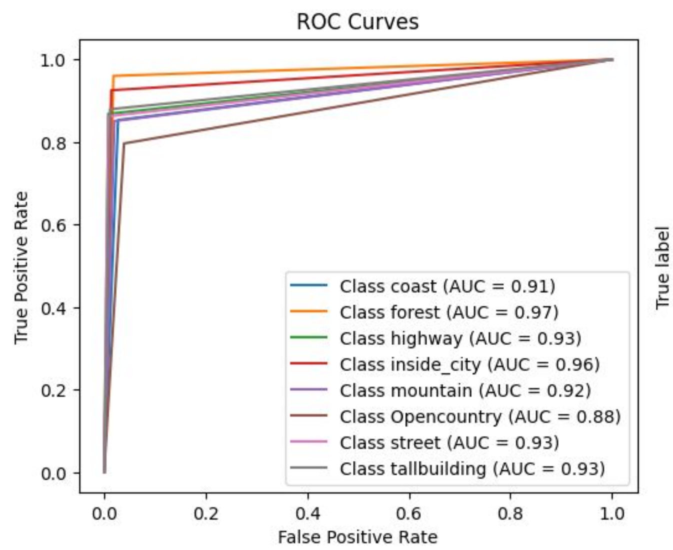
```
config = {
        'descriptor': 'dense_SIFT',
        'descriptor_param': 771,
        'k_codebook': 348,
        'step_size': 10,
        'scale_factor': 8,
        'normalize': None,
        'scale': None,
        'dim_reduction': None,
        'n_components': None,
        'model_type': 'svm',
        'C': 0.13146297809660454,
        'alpha': 0.7,
        'kernel': "histogram_intersection",
        'pyramid_level': 2
    }
```

# Test Evaluation - Quantitative Results

We compute the PR and ROC curves using a one-vs-rest strategy.

- What we observe is that both graphics present a similar result: The model performs alike along all classes, which indicates that it is not heavily biased towards a specific class.
- For better understanding the behaviour of the model, we should focus on the PR curve, since our dataset is slightly unbalanced (ROC could be misleading). The class with the lowest Average Precision is *Opencountry*, but even in this case we can see that reaches a recall of almost 80% with only 20% rate of false positives, therefore we can be sure that the model works reasonable well.



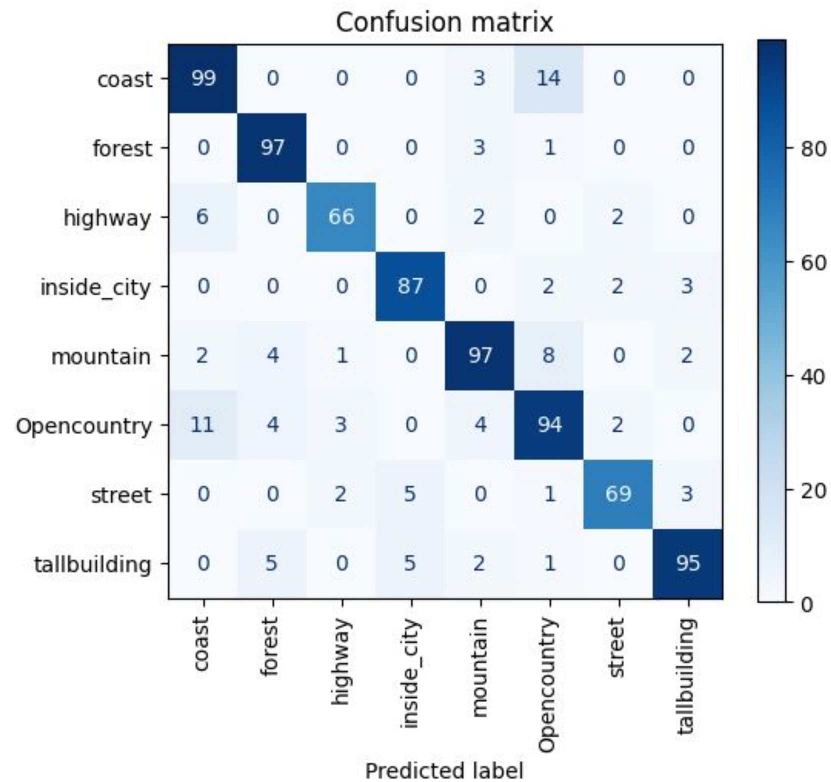
# Test Evaluation - Quantitative Results

From the confusion matrix we can conclude the following:

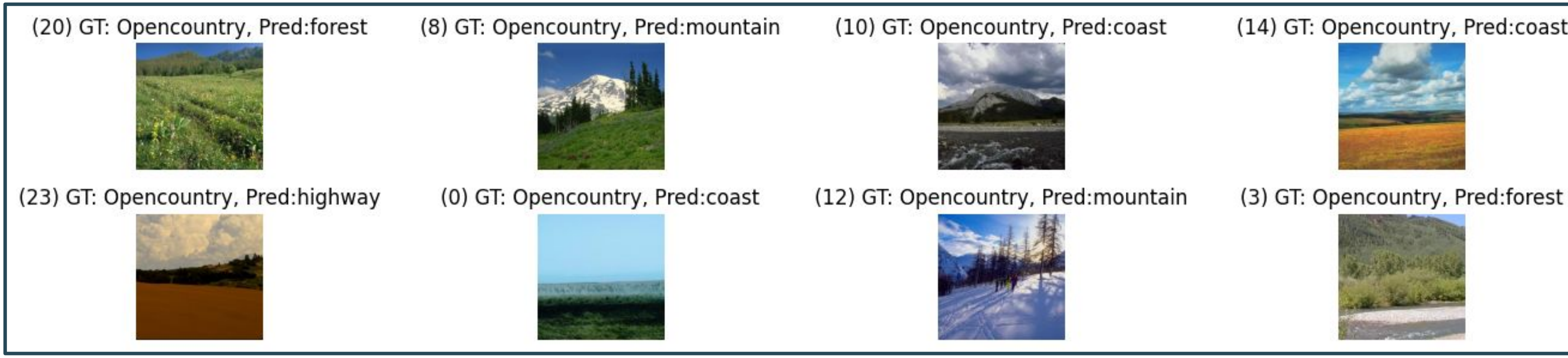
- As we have seen before, the model is able to correctly classify most of the samples → **Accuracy of 87.24%**

- *Opencountry* is the class that the model has more problems distinguishing. It is also usually predicted when a *Coast* instance is the input (the second worst performing class by AP)

- On the other hand, *Forest* seems well characterized by our features, with a very low amount of false-positives.



# Test Evaluation - Qualitative Results

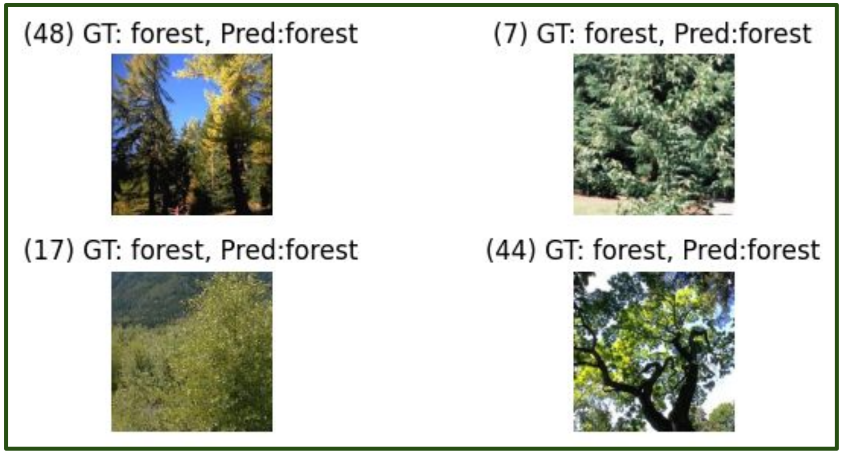


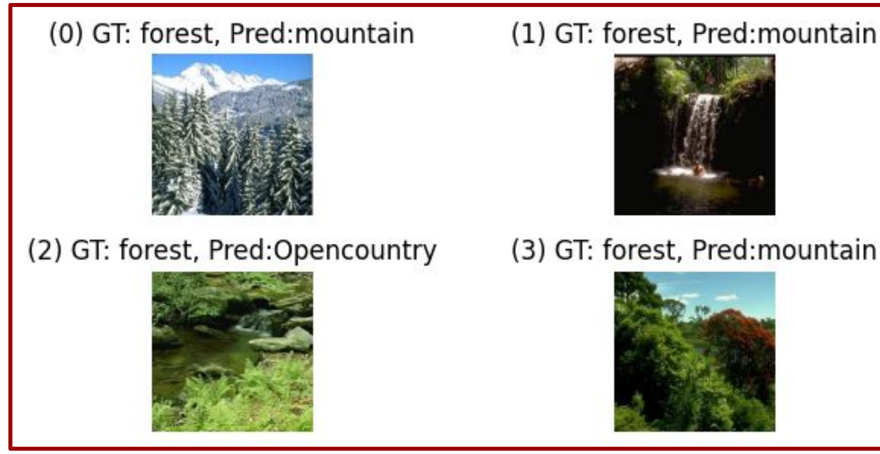

Above, we see multiple examples where the model has predicted wrongfully that the images belonged to *Opencountry*.

We can quickly realize that the actual problem with this class is that it is very ambiguous, and most of the predictions really make sense, per example when “mountain” has been predicted.

Although for this particular class this is specially obvious, this is common to most classes: we cannot expect a perfect performance in a task that even humans would fail.

# Test Evaluation - Qualitative Results




On the left, we have well predicted samples of the class *Forest*, which the model is able to characterize and classify swiftly.

On the right we present the only four samples where the model has failed, and right away we can see multiple cases of ambiguity, in the (0) and (2) examples, where the predicted label makes sense but is different to the ground truth.