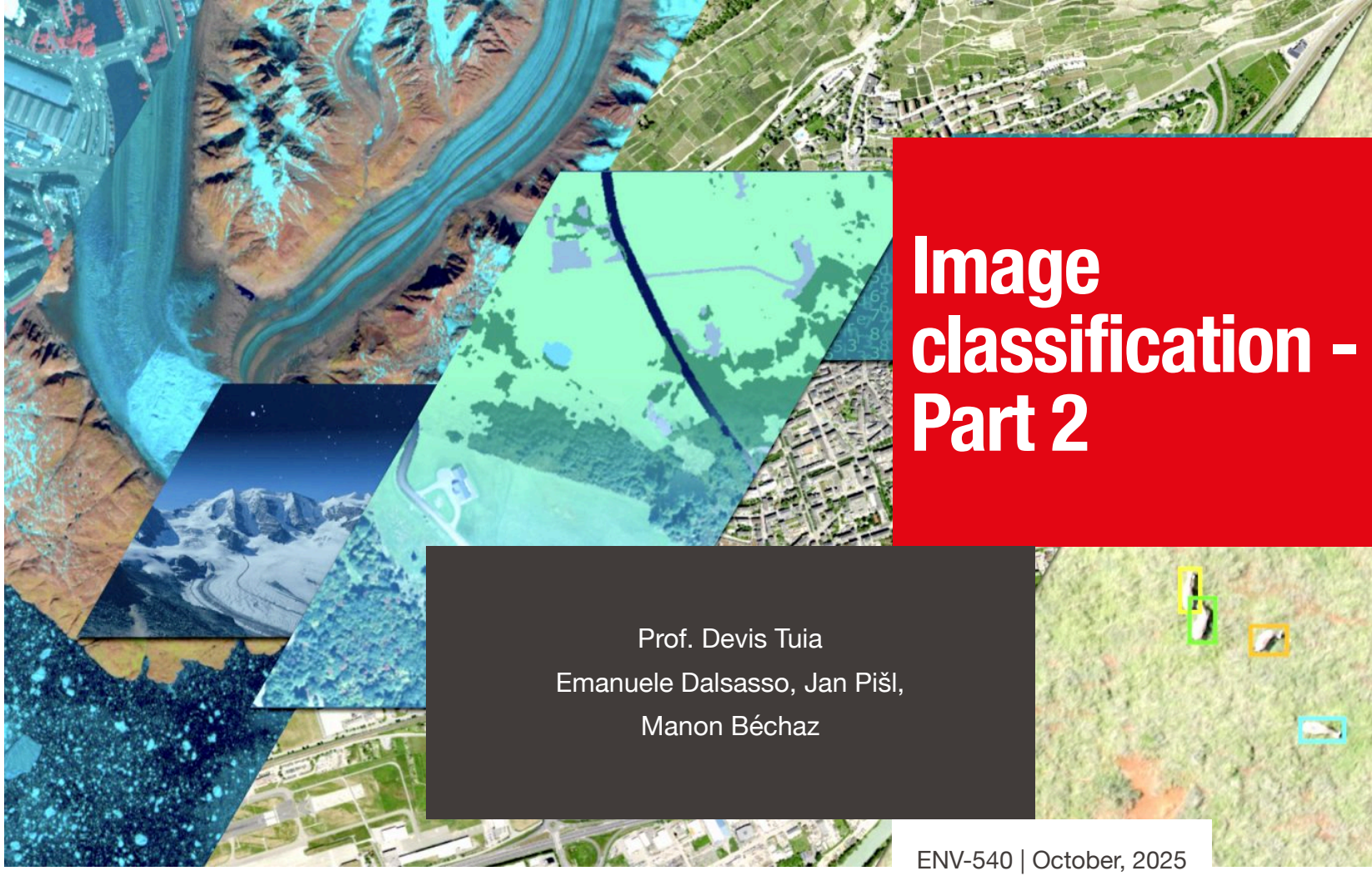
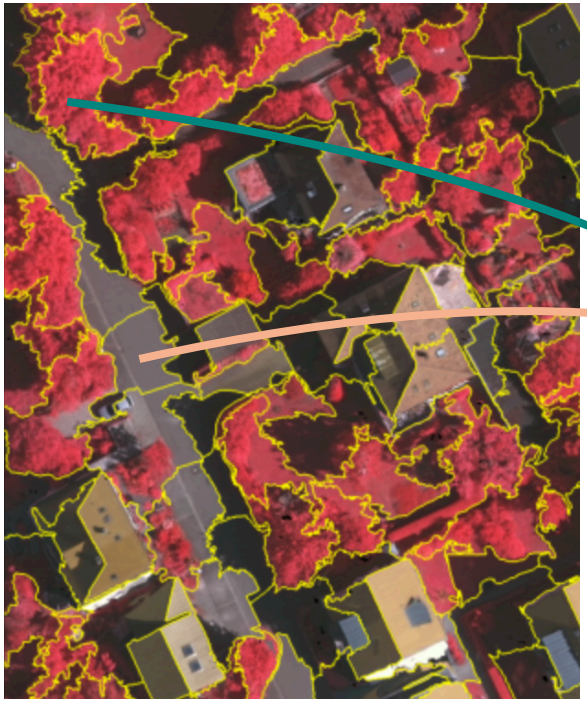


Image classification - Part 2

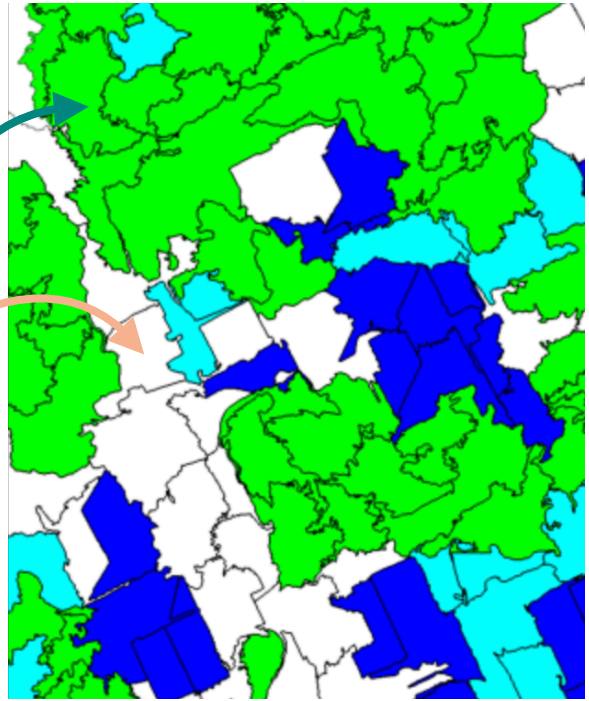
Prof. Devis Tuia
Emanuele Dalsasso, Jan Pišl,
Manon Béchaz

ENV-540 | October, 2025

Image classification



Machine Learning Model



- Impervious
- Building
- Low vegetation
- Tree

+ Parameter search on validation set
+ Accuracy of the predictions
+ Confusion matrix

Example: Split data into train, test, and validation sets


```python
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

# Load data example
wine_dataset = load_wine()
features = wine_dataset["data"]
labels = wine_dataset["target"]

# Get features and targets of the train, test and validation sets
train_features, test_features, train_labels, test_labels = train_test_split(features,
                                                                            labels,
                                                                            test_size=0.40,
                                                                            random_state=42)

train_features, val_features, train_labels, val_labels = train_test_split(train_features,
                                                                          train_labels,
                                                                          test_size=0.30,
                                                                          random_state=42)
```

Example: training a K-Nearest Neighbors (KNN) classifier


```
from sklearn.neighbors import KNeighborsClassifier

mean_array = np.mean(train_features, axis=0)
std_array = np.std(train_features, axis=0)

# Normalize features by
# substracting the mean and dividing by the standard deviation
norm_train_features = (train_features - mean_array) / std_array
norm_test_features = (test_features - mean_array) / std_array
norm_val_features = (val_features - mean_array) / std_array

# Create a K-Nearest Neighbor classifier
classifier = KNeighborsClassifier(n_neighbors=5)

# Fit model
classifier.fit(norm_train_features, train_labels)
```

Example: Computing accuracy and confusion matrix

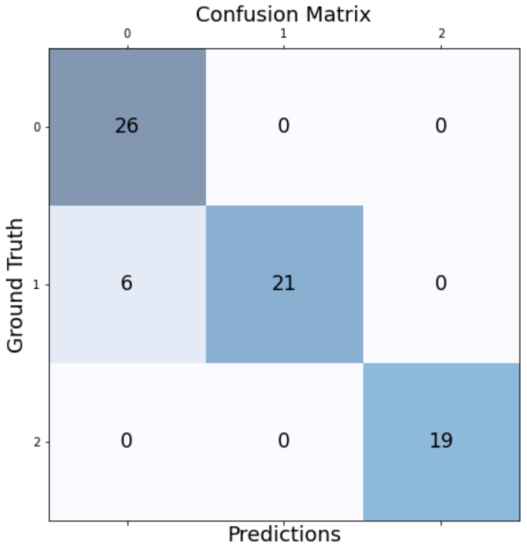

```
# Predict labels for the validation set
val_label_preds = classifier.predict(norm_val_features)

# Compute accuracy
val_acc = accuracy_score(val_labels, val_label_preds)

# Compute confusion matrix
val_conf_matrix = confusion_matrix(val_labels, val_label_preds)

# Show accuracy
print(val_acc)

# Plot confusion matrix
plot_confusion_matrix(val_conf_matrix)
```




```
0.9166666666666666
```

Parameter search on the validation set

```
# Parameter values evaluated on the validation set
n_neighbors_values = [3, 5, 7, 9, 11, 15]
# Initialize best_val_accuracy and best_n_neighbors
best_val_accuracy = -1
best_n_neighbors = None

for n_neighbors_val in n_neighbors_values:
    clf = KNeighborsClassifier(n_neighbors=n_neighbors_val)
    clf.fit(norm_train_features, train_labels)
    # Perform prediction on the validation set
    val_label_preds = classifier.predict(norm_val_features)
    val_acc = accuracy_score(val_labels, val_label_preds)
    # Change best parameter value (if it obtains better accuracy)
    if val_acc > best_val_accuracy:
        best_val_accuracy = val_acc
        best_n_neighbors = n_neighbors_val

print("best num neighbors        : " + str(best_n_neighbors))
print("best validation accuracy  : " + str(best_val_accuracy))
```

```
best num neighbors        : 3
best validation accuracy  : 0.9375
```

Train model with different parameter values and compute the accuracy on the **validation set**

Training the model with the best parameter found, and computing the accuracy on the test set

```
# Train model with the best parameter found
clf = KNeighborsClassifier(n_neighbors=best_n_neighbors)

clf.fit(norm_train_features, train_labels)

# Predict labels of the test set
preds = clf.predict(norm_test_features)

# Compute accuracy and confusion matrix
test_acc = accuracy_score(test_labels, preds)
test_cm = confusion_matrix(test_labels, preds)

# Show accuracy
print("Test accuracy : " + str(test_acc))
```

```
Test accuracy : 0.9305555555555556
```

Exercise

- Read the provided PDF file and Jupyter Notebook with detailed instructions
- Tasks:
    1. Install dependencies, if needed
    2. Extract regions and their features
    3. Create a training dataset, with the following classes
        - 0: Impervious
        - 1: Building
        - 2: Low vegetation
        - 3: Tree
    4. Normalize features
    5. Train Random Forest classifier with default parameters
    6. Predict classification maps,
    7. Compute accuracy and confusion matrix
    8. Search for good parameter values in the validation set
    9. Predict classification maps with the best parameters found in the validation set
    10. Visualize predictions
    11. Answer the questions of the PDF file of instructions