# Computervision Lab 5 - Classification

In [1]:
```
# imports
import os
from glob import glob
import cv2 as cv
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
import time

# name printing function
def print_name(im, name):
        im = cv.putText(im, name, (10, im.shape[0]-15), cv.FONT_HERSHEY_SIMPLEX, 0.
        return im
```

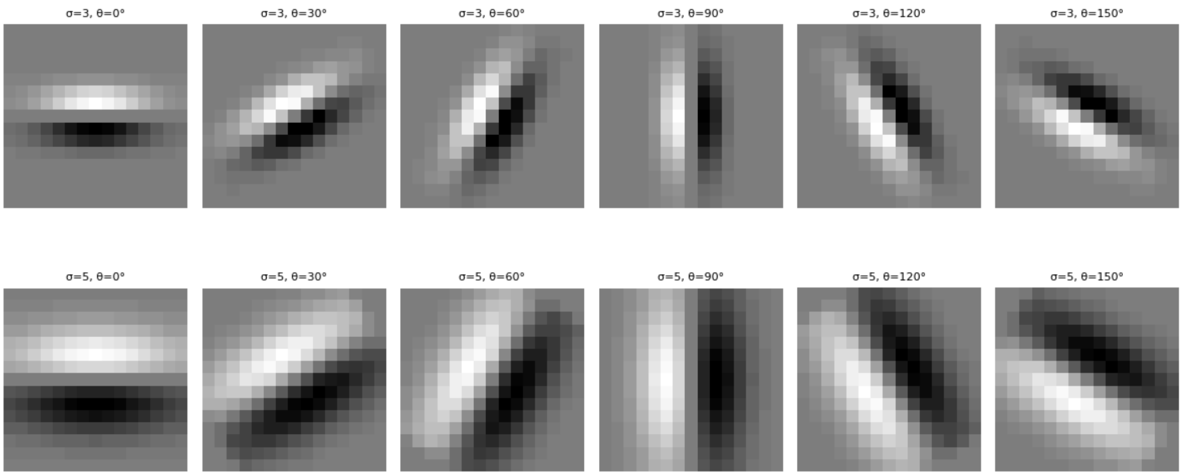
## Linear and Quadratic Discriminant Analysis

### Exercise 1

**Assignment 1:** Make a filterbank of DoG filters in 2 scales and 6 orientations (so 12 filters in total). Visualize the filters as in Figure 2.

In [2]:
```
# Ensure output directory exists
os.makedirs("out", exist_ok=True)

# Function to normalize and save filters
def save_filter(name, f):
    normalized = 0.5 * f / np.max(np.abs(f)) + 0.5  # Normalize to [0, 1]
    cv.imwrite(f"out/{name}.png", (normalized * 255).astype(np.uint8))

# Function to create DoG filter
def create_dog_filter(size, sigma1, sigma2, theta):
    """
    Generates a Difference of Gaussian (DoG) filter with a given orientation.
    """
    # Step 1: Create 1D Gaussian kernels
    gauss_kernel_1d = cv.getGaussianKernel(size, sigma1).flatten()
    gauss_kernel_1d_small = cv.getGaussianKernel(size, sigma2)

    # Step 2: Create a square matrix and insert the Gaussian as a row
    gauss_2d = np.zeros((size, size))
    gauss_2d[size // 2, :] = gauss_kernel_1d  # Horizontal Gaussian
    gauss_2d_filtered = cv.filter2D(gauss_2d, -1, gauss_kernel_1d_small)  # Smooth

    # Step 3: Compute DoG by applying Sobel
    dog_filter = cv.Sobel(gauss_2d_filtered, cv.CV_64F, 0, 1, ksize=3)  # Vertical

    # Step 4: Rotate DoG to specified angle
    center = (size // 2, size // 2)
    rotation_matrix = cv.getRotationMatrix2D(center, np.degrees(theta), 1)
    rotated_dog = cv.warpAffine(dog_filter, rotation_matrix, (size, size))
```

```python
    return rotated_dog

# Parameters
scales = [(3, 1), (5, 2)]
num_orientations = 6
size = 15  # Kernel size

# Generate and save DoG filters
filterbank = []
plt.figure(figsize=(12, 6))

for i, (sigma1, sigma2) in enumerate(scales):
    for j in range(num_orientations):
        theta = j * (np.pi / num_orientations)
        dog_filter = create_dog_filter(size, sigma1, sigma2, theta)
        filterbank.append(dog_filter)

        # Save filter
        save_filter(f"assignment1_DoG_sigma{sigma1}_theta{j*30}", dog_filter)

        # Plot
        idx = i * num_orientations + j
        plt.subplot(len(scales), num_orientations, idx + 1)
        plt.imshow(dog_filter, cmap='gray', interpolation='nearest')
        plt.axis('off')
        plt.title(f"σ={sigma1}, θ={j * 30}°", fontsize=8)

plt.tight_layout()
plt.show()
```




## Assignment 2:

- Filter road*.png with each of the filters. This gives you 12 filter response images. Make sure they are floating point and contain negative values!
- Append the 12 filter response images to the blue, green and red channels to make a 15-channel image, from which you can extract a 15-dimensional feature vector for each pixel. If you imagine the 15 channels as a stack of images lying on top of each other, each pixel's feature vector is a vertical string of values from the stack of images.
- Train and test a new QDA classifier on the 15-dimensional feature vectors of all pixels of all four images.

Show the classification result in your report.

In [3]:
```
def extract_features(image, filterbank):
    # Convert to grayscale for filter responses
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY).astype(float) / 255.0

    # Apply each filter
    filter_responses = []
    for filt in filterbank:
        # Ensure filter is float
        filt = filt.astype(float)
        # Apply filter using convolution
        response = cv.filter2D(gray, -1, filt)
        filter_responses.append(response)

    # Get color channels
    b, g, r = cv.split(image)

    # Create 15-dimensional feature vector for each pixel
    # First 3 dimensions are color channels
    # Next 12 dimensions are filter responses
    feature_vector = np.dstack([b, g, r] + filter_responses)

    # Reshape to (pixels, features)
    h, w = image.shape[:2]
    feature_vector = feature_vector.reshape(h * w, -1)

    return feature_vector

sources = sorted(glob("Images/road?.png"))
labels = sorted(glob("Images/road?_skymask.png"))

# Initialize arrays for features and labels
all_features = []
all_labels = []

# Process each image
for source, label in zip(sources, labels):
    im = cv.imread(source, 1)
    lab = cv.imread(label, 0)

    # Visualize original image with sky mask
    lab_color = cv.merge((np.zeros(lab.shape, float), (lab == 255).astype(float), (
    cv.namedWindow("input data")
    cv.imshow("input data", 0.7 * im / 255 + 0.3 * lab_color)
    cv.waitKey()
    cv.destroyWindow("input data")

    features = extract_features(im, filterbank)
    labels_flat = lab.flatten()
    all_features.append(features)
    all_labels.append(labels_flat)

# Combine features and labels from all images
features = np.vstack(all_features)
values = np.hstack(all_labels)

# Keep only pixels with value 0 or 255 (road or sky)
which = np.union1d(np.where(values == 255), np.where(values == 0))
features = features[which, :]
values = values[which]
```

In [4]:
```
# Train a QDA classifier
qda = QuadraticDiscriminantAnalysis()
qda.fit(features, values)
```

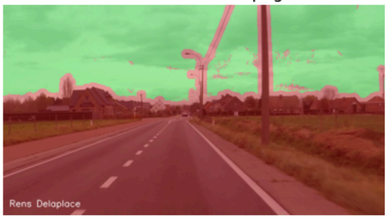
```python
print(f'Mean training accuracy: {qda.score(features, values)}')

# Prediction & Visualization
num_images = len(sources)
cols = 3  # Number of images per row
rows = (num_images + cols - 1) // cols  # Compute number of rows needed

fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))

for i, source in enumerate(sources):
    im = cv.imread(source, 1)

    # Extract features
    im_features = extract_features(im, filterbank)

    # Predict
    plab = qda.predict(im_features).reshape(im.shape[:2])

    # Create output visualization
    plab_color = cv.merge((np.zeros(plab.shape, float), (plab == 255).astype(float)
    output = (0.7 * im / 255 + 0.3 * plab_color)

    # Convert & Add Name Label
    output = (output * 255).astype(np.uint8)
    output = print_name(output, "Rens Delaplace")

    # Save Image
    output_filename = f"out/assignment2_{os.path.basename(source)}"
    cv.imwrite(output_filename, output)

    # Display in Grid Layout
    ax = axes[i // cols, i % cols] if rows > 1 else axes[i % cols]
    ax.imshow(cv.cvtColor(output, cv.COLOR_BGR2RGB))
    ax.axis("off")
    ax.set_title(f"Prediction: {os.path.basename(source)}")

# Hide empty subplots if images don't fill the grid
for j in range(i + 1, rows * cols):
    fig.delaxes(axes.flatten()[j])

plt.tight_layout()
plt.show()
```

```
C:\Rens\grind\MyPond\ImageClassification\imageclassification\Lib\site-packages\skl
earn\discriminant_analysis.py:1024: LinAlgWarning: The covariance matrix of class
0 is not full rank. Increasing the value of parameter `reg_param` might help reduc
ing the collinearity.
  warnings.warn(
C:\Rens\grind\MyPond\ImageClassification\imageclassification\Lib\site-packages\skl
earn\discriminant_analysis.py:1024: LinAlgWarning: The covariance matrix of class
1 is not full rank. Increasing the value of parameter `reg_param` might help reduc
ing the collinearity.
  warnings.warn(
Mean training accuracy: 0.9905267779462744
```

Prediction: road1.png

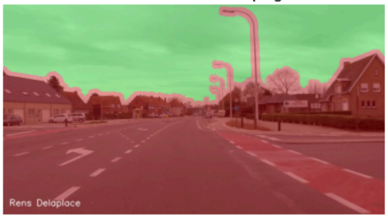

Prediction: road2.png


Prediction: road3.png


Prediction: road6.png


Prediction: road7.png


Prediction: road8.png


Prediction: road9.png


# Random forest

## Exercise 2

### **Assignment 3:** Replace the QDA classifier you trained in Exercise 14 with a random forest classifier. Pay attention to the main parameters:

In [5]:
```
# Random Forest classifier parameters
n_estimators = 30  # Number of trees in the forest
min_samples_leaf = 0.01  # Minimum samples required at a leaf node (fraction)
min_samples_leaf_abs = int(min_samples_leaf * features.shape[0])

# Train Random Forest classifier
rf = RandomForestClassifier(
    n_estimators=n_estimators,
    min_samples_leaf=min_samples_leaf_abs,
    random_state=10
)
rf.fit(features, values)
print(f'Mean training accuracy: {rf.score(features, values)}')
print(f'Number of trees: {n_estimators}')
print(f'Minimum leaf size: {min_samples_leaf_abs} samples ({min_samples_leaf*100:.2

# Prediction & Visualization
num_images = len(sources)
cols = 3  # Number of images per row
rows = (num_images + cols - 1) // cols  # number of rows

fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))
```

```python
for i, source in enumerate(sources):
    im = cv.imread(source, 1)

    # Extract features
    im_features = extract_features(im, filterbank)

    # Predict using Random Forest
    plab = rf.predict(im_features).reshape(im.shape[:2])

    # Create output visualization
    plab_color = cv.merge((np.zeros(plab.shape, float), (plab == 255).astype(float)
    output = (0.7 * im / 255 + 0.3 * plab_color)

    # Convert & Add Name Label
    output = (output * 255).astype(np.uint8)
    output = print_name(output, "Rens Delaplace")

    # Save Image
    output_filename = f"out/assignment3_{os.path.basename(source)}"
    cv.imwrite(output_filename, output)

    # Display in Grid Layout
    ax = axes[i // cols, i % cols] if rows > 1 else axes[i % cols]
    ax.imshow(cv.cvtColor(output, cv.COLOR_BGR2RGB))
    ax.axis("off")
    ax.set_title(f"Prediction: {os.path.basename(source)}")

for j in range(i + 1, rows * cols):
    fig.delaxes(axes.flatten()[j])

plt.tight_layout()
plt.show()
```

```
Mean training accuracy: 0.9980525619718955
Number of trees: 30
Minimum leaf size: 3712 samples (1.00% of training data)
```

Prediction: road1.png

Prediction: road2.png

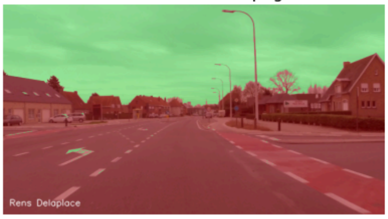
Prediction: road3.png

Prediction: road6.png

Prediction: road7.png

Prediction: road8.png

Prediction: road9.png



### **Question 1:** Does the RF classifier outperform the QDA classifier on the sky pixel classification problem?

Yes, but this comes at the cost of increased computational complexity.

# Deep Learning

## Exercise 3

### **Assignment 4:** Replace the classifier you made in Exercise 15 with a neural network. You can use the MLPClassifier from sklearn. Pay special attention to the following parameters:

- number of layers,
- number of neurons in each layer,
- learning rate of the optimizer,
- size of the training batches,
- number of training epochs.

Show the classification result in your report.

In [6]:
```
# Feature scaling
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Neural Network Parameters
```

```python
hidden_layer_sizes = (50, 25)  # Two hidden layers with 50 and 25 neurons
learning_rate_init = 0.001
batch_size = 256
max_iter = 30  # Number of epochs

print(f"Training MLP classifier with:")
print(f"- Hidden layers: {hidden_layer_sizes}")
print(f"- Learning rate: {learning_rate_init}")
print(f"- Batch size: {batch_size}")
print(f"- Max iterations: {max_iter}")

# Initialize and train the neural network
start_time = time.time()
mlp = MLPClassifier(
    hidden_layer_sizes=hidden_layer_sizes,
    learning_rate_init=learning_rate_init,
    batch_size=batch_size,
    max_iter=max_iter,
    random_state=42,
    verbose=True
)
mlp.fit(features_scaled, values)
training_time = time.time() - start_time

# Performance evaluation
print(f"Training completed in {training_time:.2f} seconds")
print(f"Training accuracy: {mlp.score(features_scaled, values)}")
print(f"Loss curve: {mlp.loss_curve_[-1]:.6f} (final loss)")

# Prediction & Visualization
num_images = len(sources)
cols = 3  # Number of images per row
rows = (num_images + cols - 1) // cols  # number of rows

fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))

for i, source in enumerate(sources):
    im = cv.imread(source, 1)

    # Extract features
    im_features = extract_features(im, filterbank)

    # Scale features
    im_features_scaled = scaler.transform(im_features)

    # Predict
    start_time = time.time()
    plab = mlp.predict(im_features_scaled)
    inference_time = time.time() - start_time
    print(f"Inference time for {source}: {inference_time:.2f} seconds")

    # Reshape prediction back to image dimensions
    plab = np.reshape(plab, (im.shape[0], im.shape[1]))

    # Create overlay visualization
    plab_color = cv.merge((np.zeros(plab.shape, float), (plab == 255).astype(float)
    output = (0.7 * im / 255 + 0.3 * plab_color)

    # Convert & Add Name Label
    output = (output * 255).astype(np.uint8)
    output = print_name(output, "Rens Delaplace")

    # Save Image
    output_filename = f"out/assignment4_{os.path.basename(source)}"
```

```
    cv.imwrite(output_filename, output)

    # Display in Grid Layout
    ax = axes[i // cols, i % cols] if rows > 1 else axes[i % cols]
    ax.imshow(cv.cvtColor(output, cv.COLOR_BGR2RGB))
    ax.axis("off")
    ax.set_title(f"Prediction: {os.path.basename(source)}")

for j in range(i + 1, rows * cols):
    fig.delaxes(axes.flatten()[j])

plt.tight_layout()
plt.show()
```

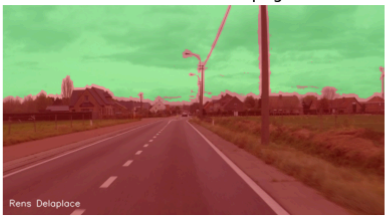
```
Training MLP classifier with:
- Hidden layers: (50, 25)
- Learning rate: 0.001
- Batch size: 256
- Max iterations: 30
Iteration 1, loss = 0.03020647
Iteration 2, loss = 0.00207047
Iteration 3, loss = 0.00152586
Iteration 4, loss = 0.00114876
Iteration 5, loss = 0.00090124
Iteration 6, loss = 0.00068069
Iteration 7, loss = 0.00057050
Iteration 8, loss = 0.00043187
Iteration 9, loss = 0.00045537
Iteration 10, loss = 0.00039259
Iteration 11, loss = 0.00035998
Iteration 12, loss = 0.00039896
Iteration 13, loss = 0.00029528
Iteration 14, loss = 0.00031004
Iteration 15, loss = 0.00027928
Iteration 16, loss = 0.00028219
Iteration 17, loss = 0.00026358
Iteration 18, loss = 0.00028171
Iteration 19, loss = 0.00025761
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. St
opping.
Training completed in 57.21 seconds
Training accuracy: 0.9999380483061598
Loss curve: 0.000258 (final loss)
Inference time for Images\road1.png: 0.20 seconds
Inference time for Images\road2.png: 0.21 seconds
Inference time for Images\road3.png: 0.19 seconds
Inference time for Images\road6.png: 0.20 seconds
Inference time for Images\road7.png: 0.19 seconds
Inference time for Images\road8.png: 0.20 seconds
Inference time for Images\road9.png: 0.19 seconds
```

Prediction: road1.png

Prediction: road2.png

Prediction: road3.png

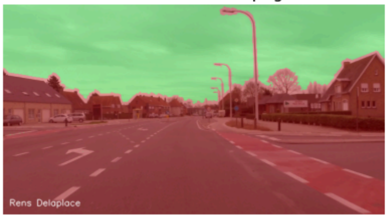
Prediction: road6.png



Prediction: road7.png

Prediction: road8.png

Prediction: road9.png



### **Question 2:** How does this classifier perform compared to the QDA classifier you made earlier? Do you see overfitting, i.e., good performance on the training data but poor performance on unseen data?

The neural network performs better than QDA overall, but shows some signs of overfitting. The training accuracy is very high compared to visible errors in prediction. The model may be memorizing training pixel patterns rather than generalizing.

### **Question 3:** Analyse the remaining errors in the prediction of your three classifiers. Where are the errors mostly located? Can you think of a simple extra feature that may help classification?

Most remaining errors are in the sky or on white cars or lines on the road. Adding a simple Y-coordinate as a feature could perhaps reduce the misclassifications.