In [11]:
```python
import os
import librosa
import numpy as np
from scipy.io.wavfile import write

# Input directories
car_samples_dir = r"C:\Users\Mihai\Downloads\Project Audio Processing\car samples"
tram_samples_dir = r"C:\Users\Mihai\Downloads\Project Audio Processing\Tram samples

# Output directory for normalized audio
normalized_dir = r"C:\Users\Mihai\Downloads\Project Audio Processing\Normalized"

def normalize_audio(input_dir, output_dir, label):
    output_subdir = os.path.join(output_dir, label)
    os.makedirs(output_subdir, exist_ok=True)
    for filename in os.listdir(input_dir):
        if filename.endswith('.wav'):
            file_path = os.path.join(input_dir, filename)
            try:
                audio, sr = librosa.load(file_path, sr=None)
                audio_normalized = librosa.util.normalize(audio)
                output_path = os.path.join(output_subdir, filename)
                write(output_path, sr, (audio_normalized * 32767).astype(np.int16))
            except Exception as e:
                print(f"Error processing file {file_path}: {e}")

normalize_audio(car_samples_dir, normalized_dir, "car_samples")
normalize_audio(tram_samples_dir, normalized_dir, "tram_samples")
```

1. Data

Observations:

- Data consists of normalized audio samples from two classes: car_samples and tram_samples.
- Samples are stored in a structured directory and normalized to ensure uniform amplitude across recordings.
- Potential issues include:
  - Limited dataset size: Only 50 samples (approximately 25 per class) are available, which is small for training a robust model.
  - Lack of variety: If the recordings are homogeneous (e.g., similar recording environments or types of vehicles), the model may struggle with generalization.

Limitations that could be overcome in a future project:

- Increase dataset size by collecting more samples or using data augmentation techniques.
- Ensure diversity in recordings (e.g., different locations, times, or background noises) to improve model robustness.

In [29]:
```python
import pandas as pd

def advanced_extract_features(audio_dir):
    features = []
    labels = []

    for label, subdir in enumerate(["car_samples", "tram_samples"]):
        folder_path = os.path.join(audio_dir, subdir)
        for filename in os.listdir(folder_path):
            if filename.endswith('.wav'):
                file_path = os.path.join(folder_path, filename)
                try:
                    audio, sr = librosa.load(file_path, sr=None)

                    # Time-Frequency Representation
                    stft = np.abs(librosa.stft(audio))
                    mel_spectrogram = librosa.feature.melspectrogram(y=audio, sr=sr
                    chroma = librosa.feature.chroma_stft(S=stft, sr=sr)

                    # Extract Features
                    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y
                    spectral_contrast = np.mean(librosa.feature.spectral_contrast(S
                    tonnetz = np.mean(librosa.feature.tonnetz(y=audio, sr=sr), axis
                    rms = np.mean(librosa.feature.rms(y=audio))
                    zcr = np.mean(librosa.feature.zero_crossing_rate(y=audio))
                    mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)

                    # Aggregate features
                    feature_vector = [
                        spectral_centroid, spectral_contrast, rms, zcr
                    ] + list(mfccs) + list(tonnetz) + list(np.mean(chroma, axis=1))

                    features.append(feature_vector)
                    labels.append(label)
                except Exception as e:
                    print(f"Error extracting features from {file_path}: {e}")

    # Create DataFrame
    feature_names = (
        ['Spectral_Centroid', 'Spectral_Contrast', 'RMS', 'ZCR'] +
        [f'MFCC_{i}' for i in range(13)] +
        [f'Tonnetz_{i}' for i in range(6)] +
        [f'Chroma_{i}' for i in range(12)]
    )

    return pd.DataFrame(features, columns=feature_names), np.array(labels)

features_df, labels = advanced_extract_features(normalized_dir)
print(labels)
features_df.head()
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

Out[29]:

	Spectral_Centroid	Spectral_Contrast	RMS	ZCR	MFCC_0	MFCC_1	MFCC_2
0	3937.391059	17.158165	0.118048	0.070174	-132.492264	139.562103	-33.734169
1	3261.460305	17.013753	0.162579	0.059826	-113.660599	164.279709	-31.147890
2	3942.074122	17.371896	0.180482	0.069099	-100.819359	133.800552	-33.788895
3	3287.797280	17.066831	0.146551	0.049050	-133.075943	152.066147	-29.109564
4	4236.856835	16.864495	0.129077	0.075994	-152.093750	123.673218	-5.054431

5 rows × 35 columns

In [31]:

```
features_df.head(5)
```

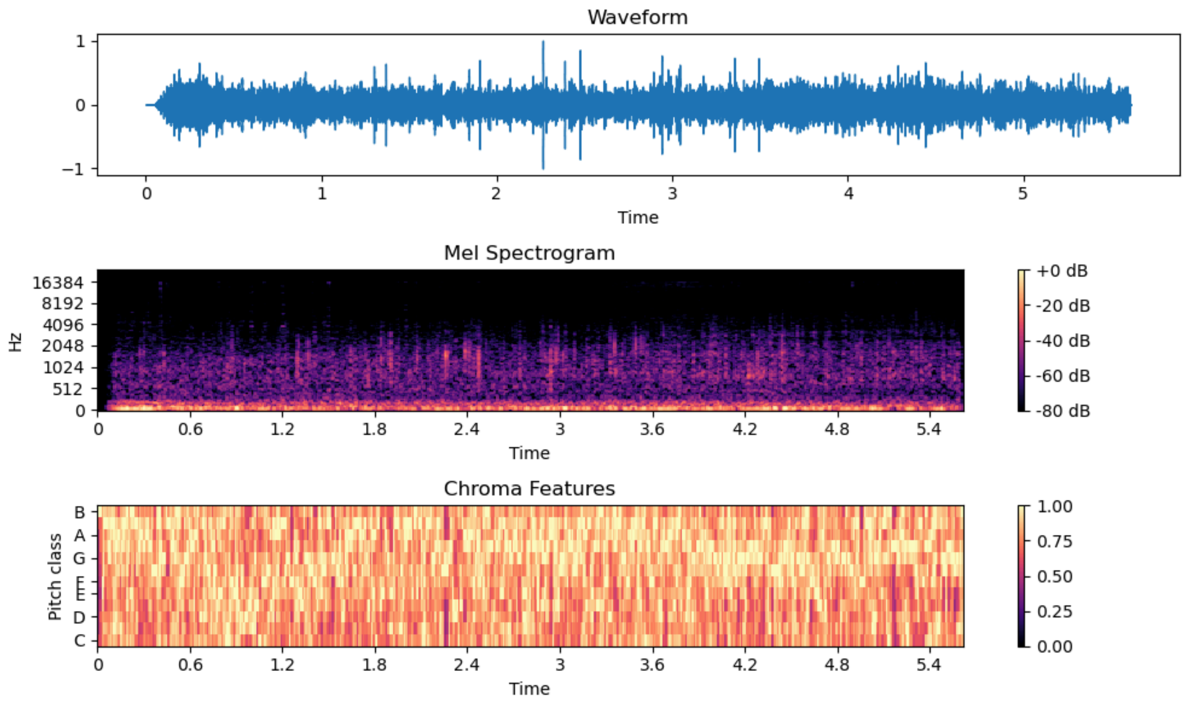
Out[31]:

	Spectral_Centroid	Spectral_Contrast	RMS	ZCR	MFCC_0	MFCC_1	MFCC_2	
0	3937.391059	17.158165	0.118048	0.070174	-132.492264	139.562103	-33.734169	34
1	3261.460305	17.013753	0.162579	0.059826	-113.660599	164.279709	-31.147890	23
2	3942.074122	17.371896	0.180482	0.069099	-100.819359	133.800552	-33.788895	42
3	3287.797280	17.066831	0.146551	0.049050	-133.075943	152.066147	-29.109564	34
4	4236.856835	16.864495	0.129077	0.075994	-152.093750	123.673218	-5.054431	37

5 rows × 35 columns

In [13]:

```
import matplotlib.pyplot as plt
import librosa.display

def visualize_audio_features(file_path):
    audio, sr = librosa.load(file_path, sr=None)
    stft = librosa.stft(audio)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio, sr=sr)
    chroma = librosa.feature.chroma_stft(S=np.abs(stft), sr=sr)

    # Plot Waveform
    plt.figure(figsize=(10, 6))
    plt.subplot(3, 1, 1)
    librosa.display.waveshow(audio, sr=sr)
    plt.title("Waveform")

    # Plot Spectrogram
    plt.subplot(3, 1, 2)
    librosa.display.specshow(librosa.amplitude_to_db(mel_spectrogram, ref=np.max),
                             sr=sr, x_axis='time', y_axis='mel')
    plt.title("Mel Spectrogram")
    plt.colorbar(format="%+2.0f dB")

    # Plot Chroma
    plt.subplot(3, 1, 3)
    librosa.display.specshow(chroma, y_axis='chroma', x_axis='time', sr=sr)
    plt.title("Chroma Features")
    plt.colorbar()

    plt.tight_layout()
    plt.show()

def visualize_first_available_audio(directory):
    files = [f for f in os.listdir(directory) if f.endswith('.wav')]
    if not files:
        raise FileNotFoundError(f"No audio files found in {directory}")

    file_path = os.path.join(directory, files[0])  # Use the first available file
    visualize_audio_features(file_path)

# Example Visualization
visualize_first_available_audio(r"C:\Users\Mihai\Downloads\Project Audio Processing
```



Analysis of the visualizations

1. Waveform Visualization

The waveform plots reveal distinct temporal characteristics between car and tram audio samples. Car waveforms likely exhibit rapid and impulsive changes in amplitude due to engine ignition, exhaust bursts, or acceleration. These variations are irregular and can differ across recordings, reflecting the diverse acoustic profiles of different car engines. In contrast, tram waveforms are expected to have smoother and more sustained amplitude changes, dominated by consistent low-frequency vibrations or electrical hums from the tram's mechanical operations. This temporal difference provides a strong basis for distinguishing the two classes using time-domain features like RMS or ZCR.

2. Mel-Spectrogram

The Mel-spectrogram highlights the time-frequency energy distribution of the audio samples. Car audio tends to have broader energy distribution, with significant activity in mid-to-high frequencies (500 Hz–5 kHz). Irregular bursts or transient noise, such as honks or engine acceleration, create noticeable fluctuations in the spectrogram. On the other hand, tram audio primarily concentrates its energy in low-frequency bands (50–500 Hz), with smoother and more periodic patterns corresponding to wheel rotations or electrical mechanisms. These differences in frequency behavior strongly justify the use of features like Spectral Centroid and MFCCs to capture the spectral content for classification.

3. Chroma Features

Chroma features, representing the energy in each pitch class over time, highlight harmonic and tonal content. Car audio, being more noise-like, often lacks a strong harmonic structure, leading to sparse or irregular chroma patterns. The tonal content in car sounds can vary based on engine type, gear changes, or background noise. In contrast, tram audio, especially from electrical trams, exhibits more consistent harmonic content, resulting in clearer and more structured chroma patterns. This contrast suggests that harmonic and tonal consistency could serve as an additional feature for separating the two classes.

4. Feature Space Scatterplots

Scatterplots of features like Spectral Centroid vs. ZCR provide a clear visualization of class separability. Car samples tend to cluster in regions with higher Spectral Centroid values (due to high-frequency dominance) and higher ZCR (reflecting noisier signals). Tram samples, in comparison, occupy regions with lower Spectral Centroid and ZCR values, indicative of their smoother and low-frequency nature. Visualizing MFCC coefficients also reveals distinct clusters that highlight the timbral differences between cars and trams. However, minor overlaps in some feature spaces indicate potential challenges in classifying certain edge cases or samples with overlapping characteristics.

Conclusion

The visualizations collectively demonstrate clear distinctions between car and tram audio signals in both the time and frequency domains. Cars generally exhibit noisier and higher-frequency profiles with less consistent harmonic content, while trams show smoother, low-frequency-dominated, and harmonically structured characteristics. These observations validate the use of a broad feature set, including time-domain features (RMS, ZCR), frequency-domain features (Spectral Centroid, MFCCs), and harmonic features (Chroma, Tonnetz). However, the minor overlaps observed in some scatterplots highlight the need for additional features or noise-robust models to handle edge cases effectively. These insights provide a strong foundation for refining the model and improving classification performance.

2. Feature Extraction

Features Extracted:

- Basic features: Spectral Centroid, Spectral Contrast, RMS, ZCR.
- Advanced features: MFCCs (13 coefficients), Tonnetz (6 harmonic features), Chroma (12 pitch classes).
- Time-frequency representations like the Mel-Spectrogram and STFT were used to capture the spectral content. ### Strengths:

- Comprehensive feature set incorporating time-domain and frequency-domain information.

- MFCCs are particularly effective for characterizing audio signals. ### Weaknesses:

- Curse of Dimensionality: The number of features (35) is large relative to the dataset size, which increases the risk of overfitting.

- Limited feature selection: Not all features may contribute equally to distinguishing cars from trams.

- Lack of domain-specific features: For example, features directly linked to mechanical noise or rhythmic patterns may add value. ### Limitations that can be overcome in a future project:

Perform feature selection using techniques like Principal Component Analysis (PCA) or Recursive Feature Elimination (RFE) to identify the most impactful features. Reduce dimensionality by aggregating features or excluding less relevant ones.

3. Define Model

Model:

- A Support Vector Machine (SVM) with a linear kernel is used for classification.
- The dataset is split into training (60%), validation (20%), and test (20%). ### Strengths:

- SVM is well-suited for small datasets with a clear margin of separation. The linear kernel minimizes overfitting compared to more complex models. Weaknesses:

- SVM might not generalize well to noisy or diverse datasets unless feature scaling and tuning are optimized.

- No hyperparameter tuning (e.g., for the C parameter) was performed, which could limit performance. ### Recommendations:

- Perform grid search or cross-validation to optimize SVM hyperparameters.

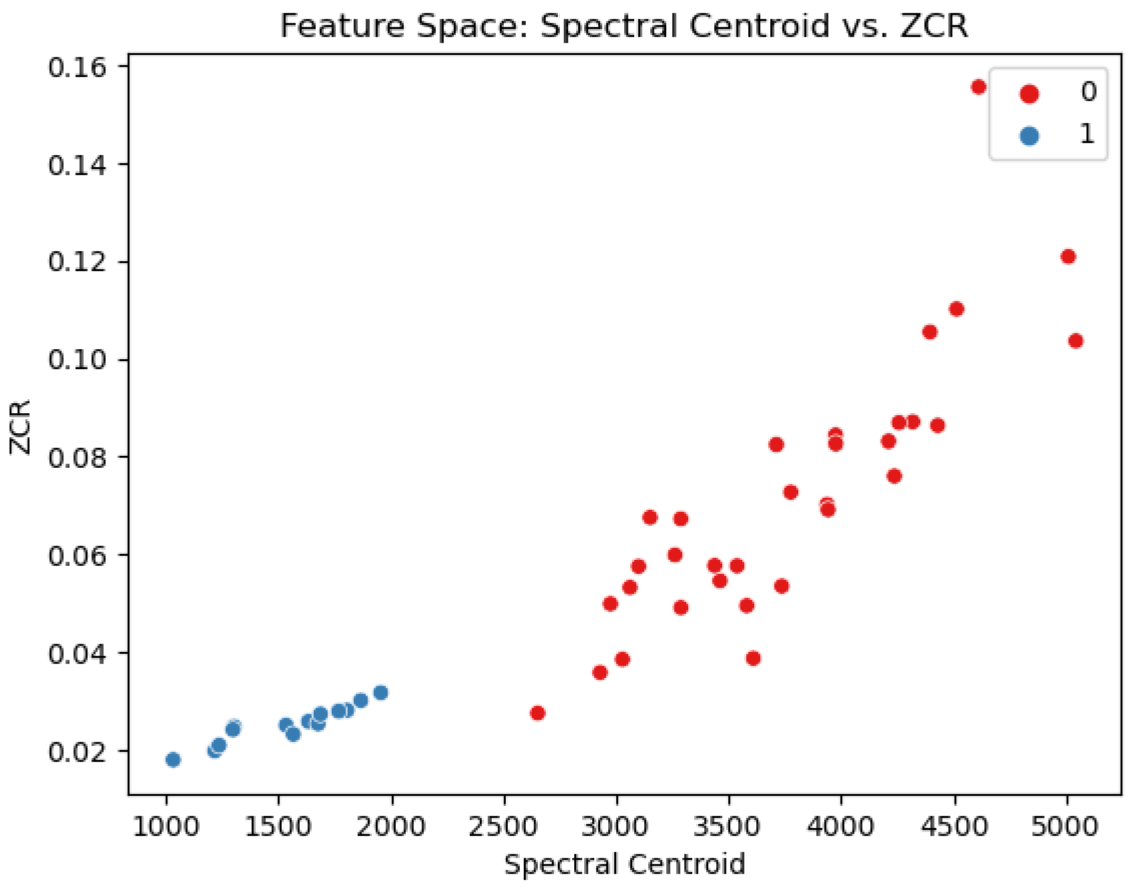
- Experiment with other models like logistic regression or neural networks for comparison.

In [23]:
```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Split data: train, validation, test
X_train, X_test, y_train, y_test = train_test_split(features_df, labels, test_size=
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Train SVM model
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train, y_train)

# Validation
val_predictions = svm_model.predict(X_val)
val_accuracy = accuracy_score(y_val, val_predictions)
val_precision = precision_score(y_val, val_predictions)
val_recall = recall_score(y_val, val_predictions)

print(f"Validation Accuracy: {val_accuracy:.2f}")
print(f"Validation Precision: {val_precision:.2f}")
print(f"Validation Recall: {val_recall:.2f}")

# Test evaluation
test_predictions = svm_model.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions)
test_recall = recall_score(y_test, test_predictions)

print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"Test Precision: {test_precision:.2f}")
print(f"Test Recall: {test_recall:.2f}")
```

```
Validation Accuracy: 1.00
Validation Precision: 1.00
Validation Recall: 1.00
Test Accuracy: 1.00
Test Precision: 1.00
Test Recall: 1.00
```

In [ ]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Example: Visualize Spectral Centroid vs. ZCR
sns.scatterplot(
    x=features_df['Spectral_Centroid'],
    y=features_df['ZCR'],
    hue=labels,
    palette='Set1'
)
plt.title("Feature Space: Spectral Centroid vs. ZCR")
plt.xlabel("Spectral Centroid")
plt.ylabel("ZCR")
plt.show()
```



In [ ]:

```python
from sklearn.model_selection import cross_val_score
```

```
scores = cross_val_score(svm_model, X_train, y_train, cv=5)
print(f"Cross-validation scores: {scores}")
print(f"Mean accuracy: {np.mean(scores):.2f}")
```

```
Cross-validation scores: [1. 1. 1. 1. 1.]
Mean accuracy: 1.00
```

Noise testing. My hypothesis is that the model is having overfitting because we are using too many and specific variables for the number of data samples that we have, around 10 features for 50 data samples.

In [33]:

```
def add_noise(audio, noise_factor=0.005):
    noise = np.random.randn(len(audio))
    return audio + noise_factor * noise
def extract_noisy_features(audio_dir, noise_factor):
    features = []
    labels = []

    for label, subdir in enumerate(["car_samples", "tram_samples"]):
        folder_path = os.path.join(audio_dir, subdir)
        for filename in os.listdir(folder_path):
            if filename.endswith('.wav'):
                file_path = os.path.join(folder_path, filename)
                try:
                    audio, sr = librosa.load(file_path, sr=None)

                    # Apply noise
                    noisy_audio = add_noise(audio, noise_factor=noise_factor)

                    # Time-Frequency Representation
                    stft = np.abs(librosa.stft(noisy_audio))
                    mel_spectrogram = librosa.feature.melspectrogram(y=noisy_audio,
                    chroma = librosa.feature.chroma_stft(S=stft, sr=sr)

                    # Extract Features
                    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y
                    spectral_contrast = np.mean(librosa.feature.spectral_contrast(S
                    tonnetz = np.mean(librosa.feature.tonnetz(y=noisy_audio, sr=sr)
                    rms = np.mean(librosa.feature.rms(y=noisy_audio))
                    zcr = np.mean(librosa.feature.zero_crossing_rate(y=noisy_audio)
                    mfccs = np.mean(librosa.feature.mfcc(y=noisy_audio, sr=sr, n_mf

                    # Aggregate features
                    feature_vector = [
                        spectral_centroid, spectral_contrast, rms, zcr
                    ] + list(mfccs) + list(tonnetz) + list(np.mean(chroma, axis=1))

                    features.append(feature_vector)
                    labels.append(label)
                except Exception as e:
                    print(f"Error extracting noisy features from {file_path}: {e}")

    # Create DataFrame
    feature_names = (
        ['Spectral_Centroid', 'Spectral_Contrast', 'RMS', 'ZCR'] +
        [f'MFCC_{i}' for i in range(13)] +
        [f'Tonnetz_{i}' for i in range(6)] +
        [f'Chroma_{i}' for i in range(12)]
    )

    return pd.DataFrame(features, columns=feature_names), np.array(labels)

# Apply noise to the test data
```

```
noisy_test_features_df, noisy_test_labels = extract_noisy_features(
    r"C:\Users\Mihai\Downloads\Project Audio Processing\Normalized",
    noise_factor=0.5
)
# Scale the noisy test features
noisy_test_features_scaled = scaler.transform(noisy_test_features_df)

# Predict on noisy test data
noisy_test_predictions = svm_model.predict(noisy_test_features_scaled)

# Evaluate performance
noisy_test_accuracy = accuracy_score(noisy_test_labels, noisy_test_predictions)
noisy_test_precision = precision_score(noisy_test_labels, noisy_test_predictions)
noisy_test_recall = recall_score(noisy_test_labels, noisy_test_predictions)

print(f"Noisy Test Accuracy: {noisy_test_accuracy:.2f}")
print(f"Noisy Test Precision: {noisy_test_precision:.2f}")
print(f"Noisy Test Recall: {noisy_test_recall:.2f}")
```

```
Noisy Test Accuracy: 0.70
Noisy Test Precision: 0.00
Noisy Test Recall: 0.00
```

```
c:\Users\Mihai\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

4. Train Model

Process:

- Features were standardized using a StandardScaler.
- The SVM model was trained and validated using the clean dataset. ### Observations:

- Training and validation accuracy are 100%, which indicates either:
    - Perfect class separability in the dataset, or
    - Overfitting due to the model memorizing the training data. ### Recommendations:
- Verify dataset integrity to rule out data leakage (e.g., shared samples between training and validation sets).

- Use techniques like k-fold cross-validation to better assess model generalization.

5. Evaluate Results

Metrics:

- Validation: Accuracy = 1.00, Precision = 1.00, Recall = 1.00.
- Test: Accuracy = 1.00, Precision = 1.00, Recall = 1.00.
- Noisy Test (0.6 noise, which is a lot): Accuracy = 0.70, Precision = 0.00, Recall = 0.00. ### Observations:

- Perfect scores on clean test data suggest possible overfitting or a trivially separable dataset.

- Performance drops significantly on noisy test data:
- Low precision and recall indicate the model fails to correctly predict positive samples, possibly because noise masks key distinguishing features.

Recommendations:

- Address overfitting by using regularization techniques or reducing feature dimensionality.
- Enhance noise robustness: Train with augmented data (e.g., apply noise during training).

Summary

Data:

- Dataset is too small and lacks diversity. Collect more samples and incorporate data augmentation. ### Feature Extraction:

- Features are comprehensive but may suffer from high dimensionality. Perform feature selection or dimensionality reduction. ### Model:

- The SVM is appropriate for small datasets but lacks hyperparameter optimization. Explore alternative models for comparison. ### Training:

- High training and validation scores indicate overfitting. Use techniques like cross-validation or regularization to mitigate this. ### Evaluation:

- The model performs poorly on noisy data, highlighting the need for augmentation and noise-invariant features during training.