DroneDetect V2 - RF-UAVNet Training

Reference Paper: Huynh-The et al. (2022), "RF-UAVNet: High-Performance Convolutional Network for RF-Based Drone Surveillance Systems", *IEEE Access*, Vol. 10, pp. 49143-49154.
DOI: 10.1109/ACCESS.2022.3173181

Overview

RF-UAVNet is a lightweight 1D CNN architecture designed for RF-based drone surveillance. This implementation adapts the IEEE paper's approach to the DroneDetect V2 dataset.

Architecture Summary

- **Input**: Raw IQ signals (2 channels: real + imaginary, 10,000 samples)
- **R-Unit**: Initial feature extraction (Conv1d 2→64, k=5, s=5)
- **G-Units**: 4x grouped convolutions (64→64, k=3, s=2, groups=8) with skip connections
- **Multi-GAP**: Multi-scale global average pooling (kernels: 1000, 500, 250, 125)
- **Classifier**: Fully connected layer (320 → num_classes)
- **Total parameters**: 9,991 (~1800x smaller than VGG16)

Dataset Comparison: DroneRF (paper) vs DroneDetect V2 (ours)

Aspect	DroneRF (paper)	DroneDetect V2 (ours)	Impact
Samples	~50,000	19,478	**2.5x fewer samples** → higher overfitting risk
Drones	10 classes	7 classes	Simpler task
Interference	Unknown	4 conditions (CLEAN/WIFI/BLUE/BOTH)	More realistic variability
Split method	Segment-level (with leakage)	File-level stratified	Scientifically valid but harder

Key trade-off: Our model has **82% fewer parameters** (9,991 vs 53M for VGG16), making it ideal for edge deployment, but our **smaller dataset** (2.5x less data) limits generalization. The paper's reported 98.53% accuracy includes **data leakage** (see RFClassification analysis); our file-level split ensures valid generalization estimates.

Hyperparameters: Aligned with IEEE Paper

Training configuration follows Huynh-The et al. (2022):

- **Optimizer**: SGD with momentum (0.95) and weight decay (1e-4)
- **Learning rate**: 0.01 (10x higher than typical Adam)
- **Batch size**: 512 (paper uses 512)
- **Epochs**: 50 (paper uses 120, reduced for computational cost)
- **Scheduler**: ReduceLROnPlateau (factor=0.5, patience=5)

These hyperparameters differ significantly from image CNNs (Adam, lr=0.001) due to RF signal characteristics.

Future Improvements

1. Transfer Learning (highest priority)

- Pre-train on DroneRF dataset (50k samples) if accessible
- Fine-tune final layers on DroneDetect V2
- Expected gain: +10-20% accuracy (mitigates small dataset issue)

2. Data Augmentation

- Time shifting (circular roll): `np.roll(iq, shift, axis=-1)`
- Noise injection (AWGN): `iq + noise`
- Expected gain: +5-10% accuracy

3. Ensemble Methods

- Combine RF-UAVNet + SVM (PSD) + VGG16 (spectrograms)
- Majority voting or stacking

- Expected gain: +3-5% accuracy

4. Segment Duration Optimization

- Test 50ms segments (paper: 10ms→20ms→50ms: 76.9%→83.6%→89.4%)
- Modify `config.DEFAULT_SEGMENT_MS` from 20 to 50

Mount Google Drive

In [ ]:
```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

In [ ]:
```
!ls drive/MyDrive/DroneDetect_V2/output/features/iq_features.npz
```

```
drive/MyDrive/DroneDetect_V2/output/features/iq_features.npz
```

Imports

In [ ]:
```
!pip install -U kaleido==0.2.1
```

```
Requirement already satisfied: kaleido==0.2.1 in /usr/local/lib/python3.12/dist-packages (0.2.1)
```

In [ ]:
```
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, precision_recall_fscore_support
from tqdm import tqdm
```

```
import os
import gc

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Setup figure saving
import re
from pathlib import Path

NOTEBOOK_NAME = "training_rfuavnet_COLAB"
FIGURES_DIR = Path("figures") / NOTEBOOK_NAME

def save_figure(fig) -> None:
    """Save plotly figure to PNG file using the figure'''s title as filename."""
    FIGURES_DIR.mkdir(parents=True, exist_ok=True)
    title = fig.layout.title.text if fig.layout.title.text else "untitled"
    filename = re.sub(r'''[^\w\s-]''', '''''', title).strip()
    filename = re.sub(r'''[\s-]+''', '''_''', filename)
    filepath = FIGURES_DIR / f"{filename}.png"
    try:
        fig.write_image(str(filepath), width=1200, height=800)
        print(f"Saved: {filepath}")
    except Exception as e:
        print(f"Warning: Could not save figure (kaleido required): {e}")
```

```
Using device: cuda
```

Configuration

In [ ]:

```
CONFIG = {
    # Paths
    'features_path': 'drive/MyDrive/DroneDetect_V2/output/features/iq_features.npz',
    'models_dir': 'drive/MyDrive/DroneDetect_V2/output/models/',
    'test_data_dir': 'drive/MyDrive/DroneDetect_V2/output/sample/test_data/',

    # Split parameters
    'test_size': 0.2,
    'random_state': 42,
```

```
    # Training parameters (aligned with Huynh-The et al. 2022, IEEE Access)
    'batch_size': 512,          # Paper: 512 (paper specification)
    'epochs': 120,               # Paper: 120 (paper specification)
    'learning_rate': 0.01,      # Paper: 0.01 (SGD with momentum)
    'momentum': 0.95,           # SGD momentum (paper specification)
    'weight_decay': 1e-4,       # L2 regularization (paper specification)
    'scheduler_factor': 0.5,    # LR reduction factor
    'scheduler_patience': 5,    # Epochs before LR reduction

    # Device
    'device': device
}

print(f"Configuration (aligned with RF-UAVNet IEEE paper): {CONFIG}")
```

```
Configuration (aligned with RF-UAVNet IEEE paper): {'features_path': 'drive/MyDrive/DroneDetect_V2/output/features/iq_features.
npz', 'models_dir': 'drive/MyDrive/DroneDetect_V2/output/models/', 'test_data_dir': 'drive/MyDrive/DroneDetect_V2/output/sampl
e/test_data/', 'test_size': 0.2, 'random_state': 42, 'batch_size': 512, 'epochs': 120, 'learning_rate': 0.01, 'momentum': 0.95,
'weight_decay': 0.0001, 'scheduler_factor': 0.5, 'scheduler_patience': 5, 'device': device(type='cuda')}
```

RF-UAV-Net Model Definition

In [ ]:

```
class RFUAVNet(nn.Module):
    """RF-UAVNet: 1D CNN architecture for RF-based drone classification.

    This model is based on the architecture proposed by Huynh-The et al. (2022) in
    "RF-UAVNet: High-Performance Convolutional Network for RF-Based Drone Surveillance Systems".
    It processes raw IQ signals (Real/Imaginary) through a series of specialized units
    (R-Unit, G-Units) and multi-scale pooling to classify drone signals.

    Attributes:
        conv_r (nn.Conv1d): Initial convolution layer (R-Unit).
        bn_r (nn.BatchNorm1d): Batch normalization for R-Unit.
        elu_r (nn.ELU): ELU activation function.
        g_convs (nn.ModuleList): List of 4 grouped convolutional layers (G-Units).
        g_bns (nn.ModuleList): List of batch normalization layers for G-Units.
        g_elus (nn.ModuleList): List of ELU activations for G-Units.
        pool (nn.MaxPool1d): Max pooling layer used in skip connections.
        gap1000 (nn.AvgPool1d): Global Average Pooling with kernel size 1000.
        gap500 (nn.AvgPool1d): Global Average Pooling with kernel size 500.
```

```
        gap250 (nn.AvgPool1d): Global Average Pooling with kernel size 250.
        gap125 (nn.AvgPool1d): Global Average Pooling with kernel size 125.
        fc (nn.Linear): Fully connected output layer.

    Args:
        num_classes (int): The number of output classes.
    """

    def __init__(self, num_classes: int):
        super().__init__()

        # R-unit
        self.conv_r = nn.Conv1d(2, 64, kernel_size=5, stride=5)
        self.bn_r = nn.BatchNorm1d(64)
        self.elu_r = nn.ELU()

        # G-units (4x)
        self.g_convs = nn.ModuleList([
            nn.Conv1d(64, 64, kernel_size=3, stride=2, groups=8)
            for _ in range(4)
        ])
        self.g_bns = nn.ModuleList([nn.BatchNorm1d(64) for _ in range(4)])
        self.g_elus = nn.ModuleList([nn.ELU() for _ in range(4)])

        self.pool = nn.MaxPool1d(kernel_size=2, stride=2)

        # Multi-scale GAP
        self.gap1000 = nn.AvgPool1d(1000)
        self.gap500 = nn.AvgPool1d(500)
        self.gap250 = nn.AvgPool1d(250)
        self.gap125 = nn.AvgPool1d(125)

        # Classifier
        self.fc = nn.Linear(320, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Forward pass of the network.

        Args:
            x (torch.Tensor): Input tensor of shape (batch_size, 2, 10000).
                Channels are [Real, Imaginary].
```

```
        Returns:
            torch.Tensor: Raw logits of shape (batch_size, num_classes).
        """
        # R-unit
        x = self.elu_r(self.bn_r(self.conv_r(x)))

        # G-units with residual connections
        g_outputs = []
        for i in range(4):
            g_out = self.g_elus[i](self.g_bns[i](self.g_convs[i](F.pad(x, (1, 0)))))
            g_outputs.append(g_out)
            x = g_out + self.pool(x)

        # Multi-scale GAP
        gaps = [
            self.gap1000(g_outputs[0]),
            self.gap500(g_outputs[1]),
            self.gap250(g_outputs[2]),
            self.gap125(g_outputs[3]),
            self.gap125(x)
        ]

        x = torch.cat(gaps, dim=1).flatten(start_dim=1)
        return self.fc(x)

    def reset_weights(self):
        """Resets the model weights using the default initialization."""
        for m in self.modules():
            if hasattr(m, 'reset_parameters'):
                m.reset_parameters()

print("RF-UAV-Net model class defined")
```

```
RF-UAV-Net model class defined
```

File-Level Stratified Split Function

In [ ]:

```
def get_stratified_file_split(X, y, file_ids, test_size=0.2, random_state=42):
    """
```

```
    Split data at FILE level to prevent data leakage.

    Segments from the same .dat file (~100 segments) will never appear
    in both train and test sets.

    Parameters
    ----------
    X : array-like
        Features (n_samples, ...)
    y : array-like
        Labels for stratification (n_samples,)
    file_ids : array-like
        Source file ID for each sample (n_samples,)
    test_size : float
        Approximate test set proportion (actual may vary due to file grouping)
    random_state : int
        Random seed for reproducibility

    Returns
    -------
    train_idx, test_idx : arrays
        Indices for train/test split
    """
    n_splits = int(1 / test_size)  # e.g., test_size=0.2 -> 5 splits -> 1 fold = 20%

    sgkf = StratifiedGroupKFold(n_splits=n_splits, shuffle=True, random_state=random_state)

    # Take first fold as train/test split
    train_idx, test_idx = next(sgkf.split(X, y, groups=file_ids))

    # Verify no file leakage
    train_files = set(file_ids[train_idx])
    test_files = set(file_ids[test_idx])
    assert len(train_files & test_files) == 0, "Data leakage detected: files in both splits"

    return train_idx, test_idx

print("Stratified file split function defined")
```

```
Stratified file split function defined
```

1. Load IQ Features

In [ ]:
```
# Pattern 1: Memory mapping to avoid loading full array into RAM
data = np.load(CONFIG['features_path'], mmap_mode='r',
               allow_pickle=True)

X = data['X']  # Shape: (N, 2, 10000) - memory mapped, not loaded
y_drone = data['y_drone']
y_interference = data['y_interference']
y_state = data['y_state']
file_ids = data['file_ids']  # For stratified file-level splitting

drone_classes = data['drone_classes']
interference_classes = data['interference_classes']
state_classes = data['state_classes']

print(f"IQ data shape: {X.shape}")
print(f"Drone labels shape: {y_drone.shape}")
print(f"File IDs shape: {file_ids.shape} (unique files: {len(np.unique(file_ids))})")
print(f"Drone classes: {drone_classes}")
print(f"Interference classes: {interference_classes}")
print(f"State classes: {state_classes}")
```

```
IQ data shape: (19478, 2, 10000)
Drone labels shape: (19478,)
File IDs shape: (19478,) (unique files: 195)
Drone classes: ['AIR' 'DIS' 'INS' 'MA1' 'MAV' 'MIN' 'PHA']
Interference classes: ['BOTH' 'CLEAN']
State classes: ['FY' 'HO' 'ON']
```

2. Train/Test Split

We'll use 80/20 split with file-level stratification to prevent data leakage.

In [ ]:
```
# Split for drone classification using file-level stratification
train_idx, test_idx = get_stratified_file_split(
    X, y_drone, file_ids,
```

```
    test_size=CONFIG['test_size'],
    random_state=CONFIG['random_state']
)

# Pattern 2: Zero-copy split (use views, not copies)
X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y_drone[train_idx], y_drone[test_idx]
y_interference_test = y_interference[test_idx]
y_state_test = y_state[test_idx]

# Verify no file leakage
train_files = set(file_ids[train_idx])
test_files = set(file_ids[test_idx])
print(f"Training files: {len(train_files)}")
print(f"Test files: {len(test_files)}")
print(f"File overlap: {len(train_files & test_files)} (should be 0)")

print(f"Training set: {X_train.shape}")
print(f"Test set: {X_test.shape}")

# Save test data for reuse
os.makedirs(CONFIG['test_data_dir'], exist_ok=True)

# Save full test data with interference and state metadata
test_data_path = os.path.join(CONFIG['test_data_dir'], 'rfuavnet_test_data.npz')
np.savez(
    test_data_path,
    X_test=X_test,
    y_test=y_test,
    y_interference_test=y_interference_test,
    y_state_test=y_state_test,
    test_idx=test_idx,
    file_ids_test=file_ids[test_idx],
    drone_classes=drone_classes,
    interference_classes=interference_classes,
    state_classes=state_classes
)
print(f"\nFull test data saved to {test_data_path}")

# Save separated files per Drone and Interference (Hierarchical)
print("\nGenerating separated test files (structure: iq/INT/DRONE/)...")
```

```
for d_idx, drone_class in enumerate(drone_classes):
    for i_idx, int_class in enumerate(interference_classes):
        # Filter for specific drone and interference
        mask = (y_test == d_idx) & (y_interference_test == i_idx)

        if not np.any(mask):
            continue

        X_sub = X_test[mask]
        y_sub = y_test[mask]
        y_int_sub = y_interference_test[mask]

        # Define components for hierarchy and filename
        data_type = 'iq'
        int_name = str(int_class)
        drone_name = str(drone_class)
        duration = '20' # 20ms fixed duration

        # Create directory structure: output/sample/test_data/{INT}/
        save_dir = os.path.join(CONFIG['test_data_dir'], int_name)
        os.makedirs(save_dir, exist_ok=True)

        # Construct filename: iq_{INT}_{DRONE}_20.npz
        filename = f"{data_type}_{int_name}_{drone_name}_{duration}.npz"
        file_path = os.path.join(save_dir, filename)

        np.savez(
            file_path,
            X=X_sub,
            y=y_sub,
            y_interference=y_int_sub,
            drone_class=drone_class,
            interference_class=int_class
        )
        print(f"  Saved {filename} in {save_dir} ({len(X_sub)} samples)")

# Cleanup: delete references to full array and indices
del X, data, train_idx, test_idx
```

```
gc.collect()
print("\nMemory cleanup: X, data, indices deleted")
```

```
Training files: 156
Test files: 39
File overlap: 0 (should be 0)
Training set: (15587, 2, 10000)
Test set: (3891, 2, 10000)

Full test data saved to drive/MyDrive/DroneDetect_V2/output/sample/test_data/rfuavnet_test_data.npz

Generating separated test files (structure: iq/INT/DRONE/)...
  Saved iq_BOTH_AIR_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (400 samples)
  Saved iq_CLEAN_AIR_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (200 samples)
  Saved iq_BOTH_DIS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (100 samples)
  Saved iq_CLEAN_DIS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (300 samples)
  Saved iq_BOTH_INS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (291 samples)
  Saved iq_CLEAN_INS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (300 samples)
  Saved iq_BOTH_MA1_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (300 samples)
  Saved iq_CLEAN_MA1_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (300 samples)
  Saved iq_BOTH_MAV_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (400 samples)
  Saved iq_CLEAN_MAV_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (300 samples)
  Saved iq_BOTH_MIN_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (300 samples)
  Saved iq_CLEAN_MIN_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (300 samples)
  Saved iq_BOTH_PHA_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH (200 samples)
  Saved iq_CLEAN_PHA_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN (200 samples)

Memory cleanup: X, data, indices deleted
```

3. Prepare PyTorch Datasets

In [ ]:
```
# Pattern 3: PyTorch conversion with immediate cleanup
# Convert train set
X_train_t = torch.from_numpy(X_train).float()
y_train_t = torch.from_numpy(y_train).long()
del X_train, y_train  # Delete numpy arrays immediately
gc.collect()

# Convert test set
X_test_t = torch.from_numpy(X_test).float()
```

```
y_test_t = torch.from_numpy(y_test).long()
del X_test, y_test  # Delete numpy arrays immediately
gc.collect()

print("PyTorch tensors created and NumPy arrays deleted")

# Create datasets
train_dataset = TensorDataset(X_train_t, y_train_t)
test_dataset = TensorDataset(X_test_t, y_test_t)

# Create dataloaders
train_loader = DataLoader(
    train_dataset,
    batch_size=CONFIG['batch_size'],
    shuffle=True
)
test_loader = DataLoader(
    test_dataset,
    batch_size=CONFIG['batch_size'],
    shuffle=False
)

print(f"Train batches: {len(train_loader)}")
print(f"Test batches: {len(test_loader)}")
```

```
PyTorch tensors created and NumPy arrays deleted
Train batches: 31
Test batches: 8
```

4. Training Function

In [ ]:

```
def train_model(model, train_loader, test_loader, config):
    """
    Train RF-UAVNet with IEEE paper hyperparameters.

    Optimizer: SGD with momentum (0.95) and weight decay (1e-4)
    Scheduler: ReduceLROnPlateau (factor=0.5, patience=5)
    """
    model = model.to(config['device'])
    criterion = nn.CrossEntropyLoss()
```

```python
# SGD optimizer (as per IEEE paper)
optimizer = optim.SGD(
    model.parameters(),
    lr=config['learning_rate'],
    momentum=config['momentum'],
    weight_decay=config['weight_decay']
)

# Learning rate scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=config['scheduler_factor'],
    patience=config['scheduler_patience'],
)

history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': [], 'lr': []}
best_val_acc = 0.0

for epoch in range(config['epochs']):
    # Training
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    for batch_x, batch_y in tqdm(train_loader, desc=f"Epoch {epoch+1}/{config['epochs']} - Train"):
        batch_x, batch_y = batch_x.to(config['device']), batch_y.to(config['device'])

        optimizer.zero_grad()
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        train_total += batch_y.size(0)
        train_correct += (predicted == batch_y).sum().item()
```

```
        train_loss /= len(train_loader)
        train_acc = train_correct / train_total

        # Validation
        model.eval()
        val_loss = 0.0
        val_correct = 0
        val_total = 0

        with torch.no_grad():
            for batch_x, batch_y in test_loader:
                batch_x, batch_y = batch_x.to(config['device']), batch_y.to(config['device'])
                outputs = model(batch_x)
                loss = criterion(outputs, batch_y)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += batch_y.size(0)
                val_correct += (predicted == batch_y).sum().item()

        val_loss /= len(test_loader)
        val_acc = val_correct / val_total

        # Learning rate scheduling
        scheduler.step(val_loss)
        current_lr = optimizer.param_groups[0]['lr']

        # Record history
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)
        history['lr'].append(current_lr)

        # Track best model
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            best_epoch = epoch + 1

        print(f"Epoch {epoch+1}: Train Loss={train_loss:.4f}, Train Acc={train_acc:.4f}, "
              f"Val Loss={val_loss:.4f}, Val Acc={val_acc:.4f}, LR={current_lr:.6f}")
```

```
        # Periodic cleanup
        if (epoch + 1) % 10 == 0:
            gc.collect()
            if torch.cuda.is_available():
                torch.cuda.empty_cache()
            print(f"  Memory cleanup at epoch {epoch+1}")

    print(f"\nBest validation accuracy: {best_val_acc:.4f} at epoch {best_epoch}")
    return model, history

print("Training function defined (SGD + ReduceLROnPlateau)")
```

```
Training function defined (SGD + ReduceLROnPlateau)
```

5. Train RF-UAV-Net

In [ ]:

```
num_classes = len(drone_classes)
rfuavnet = RFUAVNet(num_classes=num_classes)
print(rfuavnet)
```

```
RFUAVNet(
  (conv_r): Conv1d(2, 64, kernel_size=(5,), stride=(5,))
  (bn_r): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (elu_r): ELU(alpha=1.0)
  (g_convs): ModuleList(
    (0-3): 4 x Conv1d(64, 64, kernel_size=(3,), stride=(2,), groups=8)
  )
  (g_bns): ModuleList(
    (0-3): 4 x BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (g_elus): ModuleList(
    (0-3): 4 x ELU(alpha=1.0)
  )
  (pool): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (gap1000): AvgPool1d(kernel_size=(1000,), stride=(1000,), padding=(0,))
  (gap500): AvgPool1d(kernel_size=(500,), stride=(500,), padding=(0,))
  (gap250): AvgPool1d(kernel_size=(250,), stride=(250,), padding=(0,))
  (gap125): AvgPool1d(kernel_size=(125,), stride=(125,), padding=(0,))
  (fc): Linear(in_features=320, out_features=7, bias=True)
)
```

In [ ]:
```
print(f"Training RF-UAVNet with {num_classes} classes...")
print(f"Model parameters: {sum(p.numel() for p in rfuavnet.parameters()):,}")
print(f"Paper comparison: VGG16 has ~138M parameters (1800x larger)\n")

rfuavnet, history = train_model(rfuavnet, train_loader, test_loader, CONFIG)
```

```
Training RF-UAVNet with 7 classes...
Model parameters: 9,991
Paper comparison: VGG16 has ~138M parameters (1800x larger)

Epoch 1/120 - Train: 100%|██████████| 31/31 [00:02<00:00, 13.38it/s]
Epoch 1: Train Loss=1.7999, Train Acc=0.2885, Val Loss=1.9506, Val Acc=0.1822, LR=0.010000
Epoch 2/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.34it/s]
Epoch 2: Train Loss=1.6427, Train Acc=0.3337, Val Loss=2.1773, Val Acc=0.1796, LR=0.010000
Epoch 3/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.19it/s]
Epoch 3: Train Loss=1.5350, Train Acc=0.3808, Val Loss=1.8383, Val Acc=0.3112, LR=0.010000
Epoch 4/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.27it/s]
Epoch 4: Train Loss=1.4639, Train Acc=0.4156, Val Loss=1.8525, Val Acc=0.3241, LR=0.010000
Epoch 5/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
```

```
Epoch 5: Train Loss=1.4221, Train Acc=0.4479, Val Loss=2.1853, Val Acc=0.3341, LR=0.010000
Epoch 6/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.05it/s]
Epoch 6: Train Loss=1.3691, Train Acc=0.4766, Val Loss=2.1909, Val Acc=0.3374, LR=0.010000
Epoch 7/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.02it/s]
Epoch 7: Train Loss=1.3253, Train Acc=0.4932, Val Loss=1.7883, Val Acc=0.3770, LR=0.010000
Epoch 8/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.07it/s]
Epoch 8: Train Loss=1.2859, Train Acc=0.5172, Val Loss=2.2026, Val Acc=0.3022, LR=0.010000
Epoch 9/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.12it/s]
Epoch 9: Train Loss=1.2560, Train Acc=0.5219, Val Loss=1.7634, Val Acc=0.4130, LR=0.010000
Epoch 10/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.10it/s]
Epoch 10: Train Loss=1.2380, Train Acc=0.5307, Val Loss=2.4558, Val Acc=0.3166, LR=0.010000
  Memory cleanup at epoch 10
Epoch 11/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.72it/s]
Epoch 11: Train Loss=1.2032, Train Acc=0.5419, Val Loss=6.2589, Val Acc=0.2655, LR=0.010000
Epoch 12/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.80it/s]
Epoch 12: Train Loss=1.2000, Train Acc=0.5405, Val Loss=4.4071, Val Acc=0.2699, LR=0.010000
Epoch 13/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
Epoch 13: Train Loss=1.2051, Train Acc=0.5370, Val Loss=2.4927, Val Acc=0.3390, LR=0.010000
Epoch 14/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.08it/s]
Epoch 14: Train Loss=1.1428, Train Acc=0.5636, Val Loss=2.8733, Val Acc=0.3608, LR=0.010000
Epoch 15/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.05it/s]
Epoch 15: Train Loss=1.1043, Train Acc=0.5857, Val Loss=2.3293, Val Acc=0.4405, LR=0.005000
Epoch 16/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.03it/s]
Epoch 16: Train Loss=1.0658, Train Acc=0.5977, Val Loss=2.0926, Val Acc=0.4765, LR=0.005000
Epoch 17/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 17: Train Loss=1.0654, Train Acc=0.5995, Val Loss=2.3554, Val Acc=0.3590, LR=0.005000
Epoch 18/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.88it/s]
Epoch 18: Train Loss=1.0474, Train Acc=0.6002, Val Loss=3.2407, Val Acc=0.3285, LR=0.005000
Epoch 19/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 19: Train Loss=1.0436, Train Acc=0.6018, Val Loss=4.4492, Val Acc=0.2873, LR=0.005000
Epoch 20/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 20: Train Loss=1.0245, Train Acc=0.6139, Val Loss=2.5595, Val Acc=0.3274, LR=0.005000
  Memory cleanup at epoch 20
Epoch 21/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.90it/s]
Epoch 21: Train Loss=1.0203, Train Acc=0.6149, Val Loss=2.7531, Val Acc=0.3112, LR=0.002500
Epoch 22/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
```

```
Epoch 22: Train Loss=1.0021, Train Acc=0.6260, Val Loss=2.2070, Val Acc=0.4336, LR=0.002500
Epoch 23/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.93it/s]
Epoch 23: Train Loss=0.9918, Train Acc=0.6280, Val Loss=3.3926, Val Acc=0.3868, LR=0.002500
Epoch 24/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.95it/s]
Epoch 24: Train Loss=0.9821, Train Acc=0.6359, Val Loss=2.5668, Val Acc=0.3770, LR=0.002500
Epoch 25/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.07it/s]
Epoch 25: Train Loss=0.9806, Train Acc=0.6338, Val Loss=1.8197, Val Acc=0.3927, LR=0.002500
Epoch 26/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.14it/s]
Epoch 26: Train Loss=0.9744, Train Acc=0.6378, Val Loss=2.1317, Val Acc=0.2822, LR=0.002500
Epoch 27/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.69it/s]
Epoch 27: Train Loss=0.9704, Train Acc=0.6382, Val Loss=1.5468, Val Acc=0.4973, LR=0.002500
Epoch 28/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.34it/s]
Epoch 28: Train Loss=0.9602, Train Acc=0.6452, Val Loss=1.7282, Val Acc=0.4187, LR=0.002500
Epoch 29/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.24it/s]
Epoch 29: Train Loss=0.9537, Train Acc=0.6484, Val Loss=1.7361, Val Acc=0.4418, LR=0.002500
Epoch 30/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.32it/s]
Epoch 30: Train Loss=0.9477, Train Acc=0.6446, Val Loss=1.8402, Val Acc=0.4590, LR=0.002500
  Memory cleanup at epoch 30
Epoch 31/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.16it/s]
Epoch 31: Train Loss=0.9421, Train Acc=0.6522, Val Loss=2.3541, Val Acc=0.4647, LR=0.002500
Epoch 32/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.34it/s]
Epoch 32: Train Loss=0.9392, Train Acc=0.6521, Val Loss=2.4387, Val Acc=0.2781, LR=0.002500
Epoch 33/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.15it/s]
Epoch 33: Train Loss=0.9413, Train Acc=0.6475, Val Loss=2.5377, Val Acc=0.3241, LR=0.001250
Epoch 34/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.13it/s]
Epoch 34: Train Loss=0.9261, Train Acc=0.6600, Val Loss=1.9134, Val Acc=0.4883, LR=0.001250
Epoch 35/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.17it/s]
Epoch 35: Train Loss=0.9219, Train Acc=0.6584, Val Loss=2.2284, Val Acc=0.3781, LR=0.001250
Epoch 36/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.13it/s]
Epoch 36: Train Loss=0.9176, Train Acc=0.6571, Val Loss=1.5682, Val Acc=0.5148, LR=0.001250
Epoch 37/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.24it/s]
Epoch 37: Train Loss=0.9087, Train Acc=0.6670, Val Loss=1.5904, Val Acc=0.4919, LR=0.001250
Epoch 38/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.10it/s]
Epoch 38: Train Loss=0.9175, Train Acc=0.6636, Val Loss=1.7594, Val Acc=0.5050, LR=0.001250
Epoch 39/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.98it/s]
Epoch 39: Train Loss=0.9093, Train Acc=0.6670, Val Loss=2.4002, Val Acc=0.3858, LR=0.000625
```

```
Epoch 40/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 40: Train Loss=0.9069, Train Acc=0.6661, Val Loss=1.5214, Val Acc=0.5161, LR=0.000625
  Memory cleanup at epoch 40
Epoch 41/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.87it/s]
Epoch 41: Train Loss=0.9050, Train Acc=0.6685, Val Loss=1.7259, Val Acc=0.5027, LR=0.000625
Epoch 42/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.99it/s]
Epoch 42: Train Loss=0.9055, Train Acc=0.6659, Val Loss=1.5510, Val Acc=0.5125, LR=0.000625
Epoch 43/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.97it/s]
Epoch 43: Train Loss=0.8996, Train Acc=0.6711, Val Loss=1.5789, Val Acc=0.4690, LR=0.000625
Epoch 44/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.99it/s]
Epoch 44: Train Loss=0.8992, Train Acc=0.6700, Val Loss=1.6684, Val Acc=0.4816, LR=0.000625
Epoch 45/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.94it/s]
Epoch 45: Train Loss=0.9021, Train Acc=0.6688, Val Loss=1.7260, Val Acc=0.3999, LR=0.000625
Epoch 46/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.97it/s]
Epoch 46: Train Loss=0.8970, Train Acc=0.6699, Val Loss=1.5587, Val Acc=0.5120, LR=0.000313
Epoch 47/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.04it/s]
Epoch 47: Train Loss=0.8939, Train Acc=0.6724, Val Loss=1.5208, Val Acc=0.5076, LR=0.000313
Epoch 48/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.94it/s]
Epoch 48: Train Loss=0.8896, Train Acc=0.6770, Val Loss=1.5279, Val Acc=0.5112, LR=0.000313
Epoch 49/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.99it/s]
Epoch 49: Train Loss=0.8862, Train Acc=0.6748, Val Loss=1.5023, Val Acc=0.5240, LR=0.000313
Epoch 50/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.94it/s]
Epoch 50: Train Loss=0.8888, Train Acc=0.6711, Val Loss=1.5340, Val Acc=0.5420, LR=0.000313
  Memory cleanup at epoch 50
Epoch 51/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.87it/s]
Epoch 51: Train Loss=0.8890, Train Acc=0.6740, Val Loss=1.5093, Val Acc=0.5150, LR=0.000313
Epoch 52/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 52: Train Loss=0.8907, Train Acc=0.6740, Val Loss=1.6146, Val Acc=0.4955, LR=0.000313
Epoch 53/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.89it/s]
Epoch 53: Train Loss=0.8845, Train Acc=0.6766, Val Loss=1.5426, Val Acc=0.5443, LR=0.000313
Epoch 54/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.95it/s]
Epoch 54: Train Loss=0.8868, Train Acc=0.6754, Val Loss=1.5895, Val Acc=0.4834, LR=0.000313
Epoch 55/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.98it/s]
Epoch 55: Train Loss=0.8890, Train Acc=0.6735, Val Loss=1.5306, Val Acc=0.5117, LR=0.000156
Epoch 56/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.99it/s]
Epoch 56: Train Loss=0.8817, Train Acc=0.6786, Val Loss=1.5383, Val Acc=0.5132, LR=0.000156
```

```
Epoch 57/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.05it/s]
Epoch 57: Train Loss=0.8853, Train Acc=0.6745, Val Loss=1.5407, Val Acc=0.5379, LR=0.000156
Epoch 58/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.01it/s]
Epoch 58: Train Loss=0.8875, Train Acc=0.6765, Val Loss=1.5332, Val Acc=0.5058, LR=0.000156
Epoch 59/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
Epoch 59: Train Loss=0.8838, Train Acc=0.6745, Val Loss=1.5485, Val Acc=0.5369, LR=0.000156
Epoch 60/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.13it/s]
Epoch 60: Train Loss=0.8870, Train Acc=0.6767, Val Loss=1.5034, Val Acc=0.5048, LR=0.000156
  Memory cleanup at epoch 60
Epoch 61/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 61: Train Loss=0.8834, Train Acc=0.6803, Val Loss=1.5108, Val Acc=0.5361, LR=0.000078
Epoch 62/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.00it/s]
Epoch 62: Train Loss=0.8853, Train Acc=0.6793, Val Loss=1.5064, Val Acc=0.5294, LR=0.000078
Epoch 63/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.90it/s]
Epoch 63: Train Loss=0.8871, Train Acc=0.6795, Val Loss=1.4809, Val Acc=0.5276, LR=0.000078
Epoch 64/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.90it/s]
Epoch 64: Train Loss=0.8819, Train Acc=0.6771, Val Loss=1.4957, Val Acc=0.5212, LR=0.000078
Epoch 65/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 65: Train Loss=0.8802, Train Acc=0.6801, Val Loss=1.5232, Val Acc=0.5389, LR=0.000078
Epoch 66/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.05it/s]
Epoch 66: Train Loss=0.8820, Train Acc=0.6777, Val Loss=1.5150, Val Acc=0.5292, LR=0.000078
Epoch 67/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.16it/s]
Epoch 67: Train Loss=0.8781, Train Acc=0.6786, Val Loss=1.5120, Val Acc=0.5305, LR=0.000078
Epoch 68/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.17it/s]
Epoch 68: Train Loss=0.8851, Train Acc=0.6766, Val Loss=1.5049, Val Acc=0.5292, LR=0.000078
Epoch 69/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.15it/s]
Epoch 69: Train Loss=0.8815, Train Acc=0.6790, Val Loss=1.4810, Val Acc=0.5279, LR=0.000039
Epoch 70/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.18it/s]
Epoch 70: Train Loss=0.8799, Train Acc=0.6790, Val Loss=1.4942, Val Acc=0.5289, LR=0.000039
  Memory cleanup at epoch 70
Epoch 71/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.87it/s]
Epoch 71: Train Loss=0.8786, Train Acc=0.6810, Val Loss=1.4997, Val Acc=0.5261, LR=0.000039
Epoch 72/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.11it/s]
Epoch 72: Train Loss=0.8766, Train Acc=0.6817, Val Loss=1.4917, Val Acc=0.5276, LR=0.000039
Epoch 73/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.04it/s]
Epoch 73: Train Loss=0.8812, Train Acc=0.6809, Val Loss=1.5132, Val Acc=0.5240, LR=0.000039
```

```
Epoch 74/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
Epoch 74: Train Loss=0.8785, Train Acc=0.6789, Val Loss=1.5001, Val Acc=0.5325, LR=0.000039
Epoch 75/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
Epoch 75: Train Loss=0.8767, Train Acc=0.6801, Val Loss=1.4973, Val Acc=0.5279, LR=0.000020
Epoch 76/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.06it/s]
Epoch 76: Train Loss=0.8784, Train Acc=0.6781, Val Loss=1.4988, Val Acc=0.5305, LR=0.000020
Epoch 77/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.00it/s]
Epoch 77: Train Loss=0.8815, Train Acc=0.6815, Val Loss=1.5075, Val Acc=0.5341, LR=0.000020
Epoch 78/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.95it/s]
Epoch 78: Train Loss=0.8805, Train Acc=0.6799, Val Loss=1.5049, Val Acc=0.5276, LR=0.000020
Epoch 79/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.88it/s]
Epoch 79: Train Loss=0.8811, Train Acc=0.6806, Val Loss=1.5090, Val Acc=0.5279, LR=0.000020
Epoch 80/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.91it/s]
Epoch 80: Train Loss=0.8781, Train Acc=0.6806, Val Loss=1.4994, Val Acc=0.5297, LR=0.000020
  Memory cleanup at epoch 80
Epoch 81/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.80it/s]
Epoch 81: Train Loss=0.8794, Train Acc=0.6780, Val Loss=1.5068, Val Acc=0.5289, LR=0.000010
Epoch 82/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.97it/s]
Epoch 82: Train Loss=0.8811, Train Acc=0.6776, Val Loss=1.5050, Val Acc=0.5302, LR=0.000010
Epoch 83/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.05it/s]
Epoch 83: Train Loss=0.8773, Train Acc=0.6764, Val Loss=1.5093, Val Acc=0.5276, LR=0.000010
Epoch 84/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.02it/s]
Epoch 84: Train Loss=0.8844, Train Acc=0.6749, Val Loss=1.5032, Val Acc=0.5281, LR=0.000010
Epoch 85/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.11it/s]
Epoch 85: Train Loss=0.8806, Train Acc=0.6814, Val Loss=1.5039, Val Acc=0.5284, LR=0.000010
Epoch 86/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 86: Train Loss=0.8817, Train Acc=0.6776, Val Loss=1.5018, Val Acc=0.5287, LR=0.000010
Epoch 87/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.88it/s]
Epoch 87: Train Loss=0.8783, Train Acc=0.6802, Val Loss=1.4994, Val Acc=0.5325, LR=0.000005
Epoch 88/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.97it/s]
Epoch 88: Train Loss=0.8812, Train Acc=0.6784, Val Loss=1.5060, Val Acc=0.5302, LR=0.000005
Epoch 89/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.02it/s]
Epoch 89: Train Loss=0.8767, Train Acc=0.6833, Val Loss=1.5021, Val Acc=0.5281, LR=0.000005
Epoch 90/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.01it/s]
Epoch 90: Train Loss=0.8768, Train Acc=0.6814, Val Loss=1.5053, Val Acc=0.5305, LR=0.000005
  Memory cleanup at epoch 90
```

```
Epoch 91/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.94it/s]
Epoch 91: Train Loss=0.8815, Train Acc=0.6798, Val Loss=1.4991, Val Acc=0.5343, LR=0.000005
Epoch 92/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.09it/s]
Epoch 92: Train Loss=0.8823, Train Acc=0.6778, Val Loss=1.5134, Val Acc=0.5289, LR=0.000005
Epoch 93/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.93it/s]
Epoch 93: Train Loss=0.8830, Train Acc=0.6769, Val Loss=1.5085, Val Acc=0.5279, LR=0.000002
Epoch 94/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 94: Train Loss=0.8793, Train Acc=0.6804, Val Loss=1.5148, Val Acc=0.5269, LR=0.000002
Epoch 95/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 95: Train Loss=0.8793, Train Acc=0.6830, Val Loss=1.5067, Val Acc=0.5292, LR=0.000002
Epoch 96/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 96: Train Loss=0.8819, Train Acc=0.6806, Val Loss=1.4941, Val Acc=0.5320, LR=0.000002
Epoch 97/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.97it/s]
Epoch 97: Train Loss=0.8790, Train Acc=0.6806, Val Loss=1.4906, Val Acc=0.5294, LR=0.000002
Epoch 98/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.86it/s]
Epoch 98: Train Loss=0.8804, Train Acc=0.6802, Val Loss=1.5041, Val Acc=0.5299, LR=0.000002
Epoch 99/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.79it/s]
Epoch 99: Train Loss=0.8832, Train Acc=0.6749, Val Loss=1.5174, Val Acc=0.5263, LR=0.000001
Epoch 100/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.93it/s]
Epoch 100: Train Loss=0.8780, Train Acc=0.6810, Val Loss=1.5058, Val Acc=0.5276, LR=0.000001
  Memory cleanup at epoch 100
Epoch 101/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.85it/s]
Epoch 101: Train Loss=0.8812, Train Acc=0.6802, Val Loss=1.5050, Val Acc=0.5287, LR=0.000001
Epoch 102/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.01it/s]
Epoch 102: Train Loss=0.8780, Train Acc=0.6767, Val Loss=1.4977, Val Acc=0.5289, LR=0.000001
Epoch 103/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 103: Train Loss=0.8792, Train Acc=0.6795, Val Loss=1.5026, Val Acc=0.5294, LR=0.000001
Epoch 104/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.88it/s]
Epoch 104: Train Loss=0.8791, Train Acc=0.6783, Val Loss=1.5050, Val Acc=0.5271, LR=0.000001
Epoch 105/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.82it/s]
Epoch 105: Train Loss=0.8813, Train Acc=0.6797, Val Loss=1.5262, Val Acc=0.5256, LR=0.000001
Epoch 106/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.85it/s]
Epoch 106: Train Loss=0.8815, Train Acc=0.6782, Val Loss=1.5014, Val Acc=0.5292, LR=0.000001
Epoch 107/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.86it/s]
Epoch 107: Train Loss=0.8799, Train Acc=0.6823, Val Loss=1.5153, Val Acc=0.5274, LR=0.000001
```

```
Epoch 108/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.86it/s]
Epoch 108: Train Loss=0.8788, Train Acc=0.6815, Val Loss=1.5178, Val Acc=0.5284, LR=0.000001
Epoch 109/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.88it/s]
Epoch 109: Train Loss=0.8763, Train Acc=0.6790, Val Loss=1.5088, Val Acc=0.5323, LR=0.000001
Epoch 110/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.80it/s]
Epoch 110: Train Loss=0.8779, Train Acc=0.6798, Val Loss=1.5070, Val Acc=0.5307, LR=0.000001
  Memory cleanup at epoch 110
Epoch 111/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.75it/s]
Epoch 111: Train Loss=0.8802, Train Acc=0.6810, Val Loss=1.4907, Val Acc=0.5323, LR=0.000000
Epoch 112/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.90it/s]
Epoch 112: Train Loss=0.8789, Train Acc=0.6770, Val Loss=1.4973, Val Acc=0.5297, LR=0.000000
Epoch 113/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.91it/s]
Epoch 113: Train Loss=0.8843, Train Acc=0.6776, Val Loss=1.5077, Val Acc=0.5320, LR=0.000000
Epoch 114/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.85it/s]
Epoch 114: Train Loss=0.8837, Train Acc=0.6792, Val Loss=1.5091, Val Acc=0.5269, LR=0.000000
Epoch 115/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.03it/s]
Epoch 115: Train Loss=0.8841, Train Acc=0.6787, Val Loss=1.5003, Val Acc=0.5261, LR=0.000000
Epoch 116/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.98it/s]
Epoch 116: Train Loss=0.8830, Train Acc=0.6780, Val Loss=1.5115, Val Acc=0.5266, LR=0.000000
Epoch 117/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 117: Train Loss=0.8818, Train Acc=0.6767, Val Loss=1.5054, Val Acc=0.5292, LR=0.000000
Epoch 118/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 16.00it/s]
Epoch 118: Train Loss=0.8783, Train Acc=0.6810, Val Loss=1.5090, Val Acc=0.5281, LR=0.000000
Epoch 119/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.92it/s]
Epoch 119: Train Loss=0.8809, Train Acc=0.6749, Val Loss=1.5032, Val Acc=0.5315, LR=0.000000
Epoch 120/120 - Train: 100%|██████████| 31/31 [00:01<00:00, 15.96it/s]
Epoch 120: Train Loss=0.8823, Train Acc=0.6802, Val Loss=1.5128, Val Acc=0.5289, LR=0.000000
  Memory cleanup at epoch 120

Best validation accuracy: 0.5443 at epoch 53
```

6. Plot Training History

In [ ]:

```python
# Training history visualization with plotly
fig = make_subplots(rows=1, cols=2, subplot_titles=('RF-UAVNet Loss', 'RF-UAVNet Accuracy'))
```

```python
epochs = list(range(1, len(history['train_loss']) + 1))

# Loss subplot
fig.add_trace(go.Scatter(x=epochs, y=history['train_loss'], mode='lines+markers', name='Train Loss', line=dict(color='blue')),
fig.add_trace(go.Scatter(x=epochs, y=history['val_loss'], mode='lines+markers', name='Val Loss', line=dict(color='orange')), r

# Accuracy subplot
fig.add_trace(go.Scatter(x=epochs, y=history['train_acc'], mode='lines+markers', name='Train Acc', line=dict(color='blue')), r
fig.add_trace(go.Scatter(x=epochs, y=history['val_acc'], mode='lines+markers', name='Val Acc', line=dict(color='orange')), row

fig.update_layout(
    title='RF-UAVNet Training History',
    height=500,
    width=1200
)
fig.update_xaxes(title_text='Epoch', row=1, col=1)
fig.update_yaxes(title_text='Loss', row=1, col=1)
fig.update_xaxes(title_text='Epoch', row=1, col=2)
fig.update_yaxes(title_text='Accuracy', row=1, col=2)

fig.show()
save_figure(fig)
```

Saved: figures/training_rfuavnet_COLAB/RF_UAVNet_Training_History.png

7. Evaluate on Test Set

In [ ]:
```
def evaluate_model(model, test_loader):
    """
    Evaluate model on test set with efficient memory usage.
    """
```

```
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_x = batch_x.to(CONFIG['device'])
            outputs = model(batch_x)
            _, predicted = torch.max(outputs, 1)

            # Pattern 6: Append to lists directly (avoid unnecessary copies)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(batch_y.numpy())

    return np.array(all_preds), np.array(all_labels)

preds, labels = evaluate_model(rfuavnet, test_loader)
accuracy = accuracy_score(labels, preds)
f1 = f1_score(labels, preds, average='weighted')

print(f"RF-UAV-Net Test Accuracy: {accuracy:.4f}")
print(f"RF-UAV-Net Test F1-Score (weighted): {f1:.4f}")
print("Classification Report:")
print(classification_report(labels, preds, target_names=drone_classes))
```

```
RF-UAV-Net Test Accuracy: 0.5289
RF-UAV-Net Test F1-Score (weighted): 0.5239
Classification Report:
              precision    recall  f1-score   support

         AIR       0.36      0.46      0.40       600
         DIS       0.89      0.59      0.71       400
         INS       0.42      0.53      0.47       591
         MA1       0.19      0.16      0.17       600
         MAV       0.62      0.37      0.46       700
         MIN       0.87      0.92      0.90       600
         PHA       0.58      0.82      0.68       400

    accuracy                           0.53      3891
   macro avg       0.56      0.55      0.54      3891
weighted avg       0.55      0.53      0.52      3891
```

8. Confusion Matrix

In [ ]:

```python
cm = confusion_matrix(labels, preds)

# Create confusion matrix heatmap with plotly
fig = go.Figure(data=go.Heatmap(
    z=cm,
    x=list(drone_classes),
    y=list(drone_classes),
    colorscale='Purples',
    text=cm,
    texttemplate='%{text}',
    textfont={'size': 12},
    hoverongaps=False
))

fig.update_layout(
    title=f'RF-UAVNet Confusion Matrix - Accuracy: {accuracy:.4f}',
    xaxis_title='Predicted',
    yaxis_title='True',
    xaxis={'side': 'bottom'},
```

```
    yaxis={'autorange': 'reversed'},
    width=800,
    height=700
)
fig.show()
save_figure(fig)
```

```
Saved: figures/training_rfuavnet_COLAB/RF_UAVNet_Confusion_Matrix_Accuracy_05289.png
```

9. Per-Class Performance

In [ ]:

```python
# Calculate per-class metrics
precision, recall, f1_per_class, support = precision_recall_fscore_support(
    labels, preds, labels=range(len(drone_classes)), zero_division=0
)

import pandas as pd

# Create DataFrame for display
metrics_df = pd.DataFrame({
    'Class': drone_classes,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1_per_class,
    'Support': support
})

print("\nPer-Class Performance:")
print(metrics_df.to_string(index=False))

# Precision plot
fig_precision = px.bar(metrics_df, x='Class', y='Precision', title='RF-UAVNet Precision per Class',
                       color='Precision', range_y=[0, 1.05])
fig_precision.update_layout(xaxis_title="Class", yaxis_title="Precision", height=400)
fig_precision.show()
save_figure(fig_precision)

# Recall plot
fig_recall = px.bar(metrics_df, x='Class', y='Recall', title='RF-UAVNet Recall per Class',
                    color='Recall', color_continuous_scale=px.colors.sequential.Oranges, range_y=[0, 1.05])
fig_recall.update_layout(xaxis_title="Class", yaxis_title="Recall", height=400)
fig_recall.show()
save_figure(fig_recall)

# F1-Score plot
fig_f1 = px.bar(metrics_df, x='Class', y='F1-Score', title='RF-UAVNet F1-Score per Class',
```

```
                color='F1-Score', color_continuous_scale=px.colors.sequential.Greens, range_y=[0, 1.05])
fig_f1.update_layout(xaxis_title="Class", yaxis_title="F1-Score", height=400)
fig_f1.show()
save_figure(fig_f1)
```

```
Per-Class Performance:
Class  Precision   Recall  F1-Score  Support
  AIR   0.358344 0.461667  0.403496      600
  DIS   0.890977 0.592500  0.711712      400
  INS   0.421409 0.526227  0.468021      591
  MA1   0.190763 0.158333  0.173042      600
  MAV   0.618357 0.365714  0.459605      700
  MIN   0.870866 0.921667  0.895547      600
  PHA   0.580247 0.822500  0.680455      400
```

```
Saved: figures/training_rfuavnet_COLAB/RF_UAVNet_Precision_per_Class.png
```

Saved: figures/training_rfuavnet_COLAB/RF_UAVNet_Recall_per_Class.png

```
Saved: figures/training_rfuavnet_COLAB/RF_UAVNet_F1_Score_per_Class.png
```

10. Save Model

In [ ]:
```python
# Ensure the directory exists
os.makedirs(CONFIG['models_dir'], exist_ok=True)

model_path = os.path.join(CONFIG['models_dir'], 'rfuavnet_iq.pth')

torch.save({
    'model_state_dict': rfuavnet.state_dict(),
    'classes': drone_classes,
    'accuracy': accuracy,
    'f1': f1,
    'history': history,
```

```
    'config': CONFIG
}, model_path)

print(f"Model saved to {model_path}")
```

```
Model saved to drive/MyDrive/DroneDetect_V2/output/models/rfuavnet_iq.pth
```

12. Summary

Key takeaways:

- RF-UAV-Net processes raw IQ data (2 channels, 10000 samples)
- Architecture: R-Unit (Conv1d 2->64) + 4 G-Units (grouped convolutions)
- Multi-scale GAP for feature aggregation
- File-level stratified split prevents data leakage
- Lower learning rate (0.001) compared to CNN (0.0001)

In [ ]:
```
print("=== RF-UAV-Net Training Summary ===")
print(f"Dataset:")
print(f"  Total samples: {len(X_train_t) + len(X_test_t)}")
print(f"  Training samples: {len(X_train_t)}")
print(f"  Test samples: {len(X_test_t)}")
print(f"  Number of classes: {num_classes}")

print(f"Training Configuration:")
print(f"  Batch size: {CONFIG['batch_size']}")
print(f"  Epochs: {CONFIG['epochs']}")
print(f"  Learning rate: {CONFIG['learning_rate']}")
print(f"  Device: {CONFIG['device']}")

print(f"Performance:")
print(f"  Test Accuracy: {accuracy:.4f}")
print(f"  Test F1-Score: {f1:.4f}")
print(f"  Final Train Loss: {history['train_loss'][-1]:.4f}")
print(f"  Final Val Loss: {history['val_loss'][-1]:.4f}")

print(f"Model saved to: {model_path}")
```

```
=== RF-UAV-Net Training Summary ===
Dataset:
  Total samples: 19478
  Training samples: 15587
  Test samples: 3891
  Number of classes: 7
Training Configuration:
  Batch size: 512
  Epochs: 120
  Learning rate: 0.01
  Device: cuda
Performance:
  Test Accuracy: 0.5289
  Test F1-Score: 0.5239
  Final Train Loss: 0.8823
  Final Val Loss: 1.5128
Model saved to: drive/MyDrive/DroneDetect_V2/output/models/rfuavnet_iq.pth
```