1. Mount the UAV Dataset from Google Drive.

```
# Now we will run the code again addressing some serious issues we faced last time.
from google.colab import drive
import os
import mat73
import numpy as np

# Mount Google Drive
drive.mount('/content/drive', force_remount=True)

# Base path to your dataset (update if needed)
base_path = '/content/drive/MyDrive/Project Work/URC drones dataset v1/'

# Load flattened spectrograms & labels from .mat files
train_data  = mat73.loadmat(os.path.join(base_path, 'URC_drones_S_dB_flat_train_dataset_v1.mat'))
test_data   = mat73.loadmat(os.path.join(base_path, 'URC_drones_S_dB_flat_test_dataset_v1.mat'))
train_label = mat73.loadmat(os.path.join(base_path, 'URC_drones_y_labels_train_dataset_v1.mat'))
test_label  = mat73.loadmat(os.path.join(base_path, 'URC_drones_y_labels_test_dataset_v1.mat'))

# Extract arrays and fix label range
X_train_1 = train_data['S_dB_flat']
X_test_1  = test_data['S_dB_flat']
y_train_1 = train_label['y_label'] - 1
y_test_1  = test_label['y_label'] - 1
```

```
Mounted at /content/drive
```

2. Shuffle the Training Dataset

```
# Now we will shuffle the dataset

def shuffle(X, y):
    rgen = np.random.RandomState(0)
    r = rgen.permutation(len(y))
    return X[r], y[r]

X_train_1, y_train_1 = shuffle(X_train_1, y_train_1)
```

3. Nornamilze the Dataset

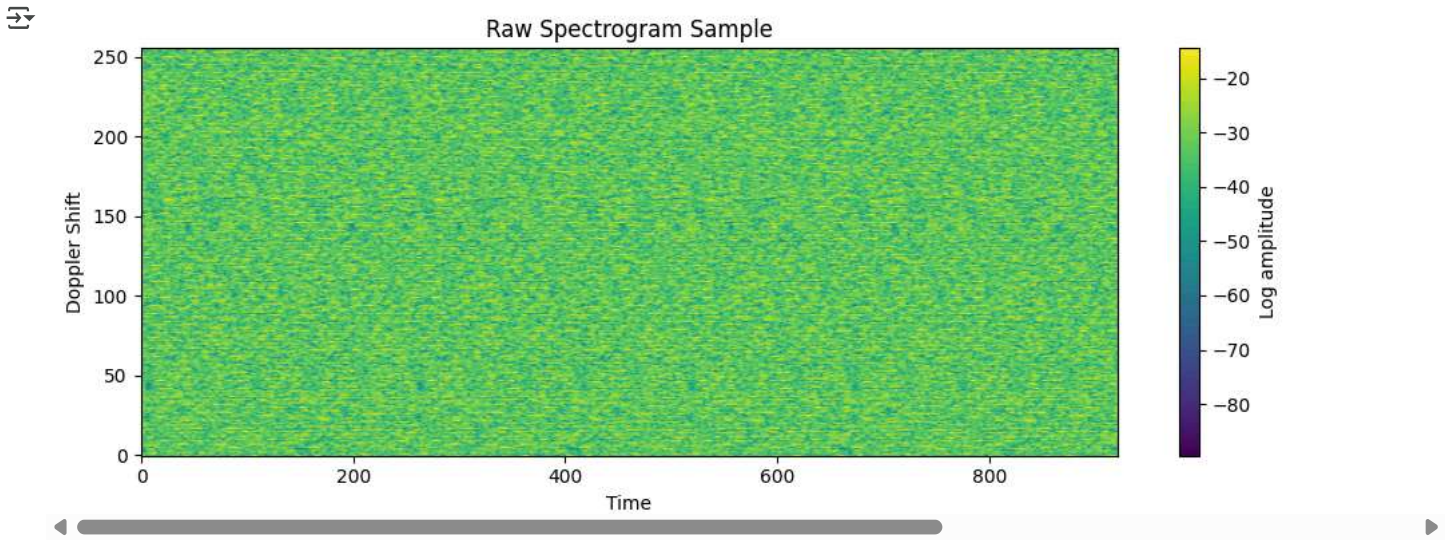
```
# We will Normalize the spectrograms PROPERLY

def normalize_signature(sig_1d, height=256, width=922):
    sig_2d = sig_1d.reshape(height, width)
    col_means = np.mean(sig_2d, axis=0)  # ✅ Column-wise mean
    sig_centered = sig_2d - col_means # ✅ Subtract column mean
    global_std = np.std(sig_2d)  # ✅ Global standard deviation of 2D matri
    if global_std == 0:
        global_std = 1e-6 # ✅ Prevent division by zero
    return sig_centered / global_std # ✅ Normalize with global std

X_train_norm = np.array([normalize_signature(sig) for sig in X_train_1])
X_test_norm = np.array([normalize_signature(sig) for sig in X_test_1])

# Confirming shape after normalization
print("X_train_norm shape:", X_train_norm.shape)  # Should be (samples, 256, 922)

# Reshape for CNN input
X_train = X_train_norm[..., np.newaxis]
X_test = X_test_norm[..., np.newaxis]
```

```
X_train_norm shape: (5600, 256, 922)
```

4. Printing Random log-domain spectogram

```
import matplotlib.pyplot as plt
```

```
def plot_spectrogram(sig_1d, height=256, width=922, title="Spectrogram"):
    sig_2d = sig_1d.reshape(height, width)
    plt.figure(figsize=(10, 4))
    plt.imshow(sig_2d, aspect='auto', origin='lower', cmap='viridis')
    plt.colorbar(label="Log amplitude")
    plt.title(title)
    plt.xlabel("Time")
    plt.ylabel("Doppler Shift")
    plt.tight_layout()
    plt.show()

# Example usage:
plot_spectrogram(X_train_1[5000], title="Raw Spectrogram Sample")
```




```
# One-Hot Encode Labels

from sklearn.preprocessing import OneHotEncoder

# ========== ONE-HOT ENCODING WITH SAFETY CHECK ==========
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
y_train = encoder.fit_transform(y_train_1.reshape(-1, 1))
y_test = encoder.transform(y_test_1.reshape(-1, 1))

print("✅ One-hot encoding complete. Shape:", y_train.shape)
print("Sample inverse:", encoder.inverse_transform(y_train[:1]))
```

```
✅ One-hot encoding complete. Shape: (5600, 5)
Sample inverse: [[0.]]
```

```
# Setting up GPU Strategy

import tensorflow as tf

# Detect and assign strategy
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    strategy = tf.distribute.MirroredStrategy()
    print("✅ GPU found, using MirroredStrategy.")
else:
    strategy = tf.distribute.get_strategy()
    print("⚠️ Using CPU only.")

print("Number of devices:", strategy.num_replicas_in_sync)
```

```
✅ GPU found, using MirroredStrategy.
Number of devices: 1
```

```
# Now let us work with more complex model to improve even more

import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, InputLayer
```

```
from tensorflow.keras.optimizers import Adam

# Build the correct model
model = Sequential()

# 1st Convolutional Layer
model.add(InputLayer(input_shape=(256, 922, 1)))
model.add(Conv2D(6, kernel_size=(5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 2nd Convolutional Layer
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Flatten & Fully Connected Layers
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))

# Output Layer for 5 UAV classes
model.add(Dense(5, activation='softmax'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27: UserWarning: Argument `input_shape` is deprecated. Use
  warnings.warn(
```

```
model.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])


model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_test, y_test))
```

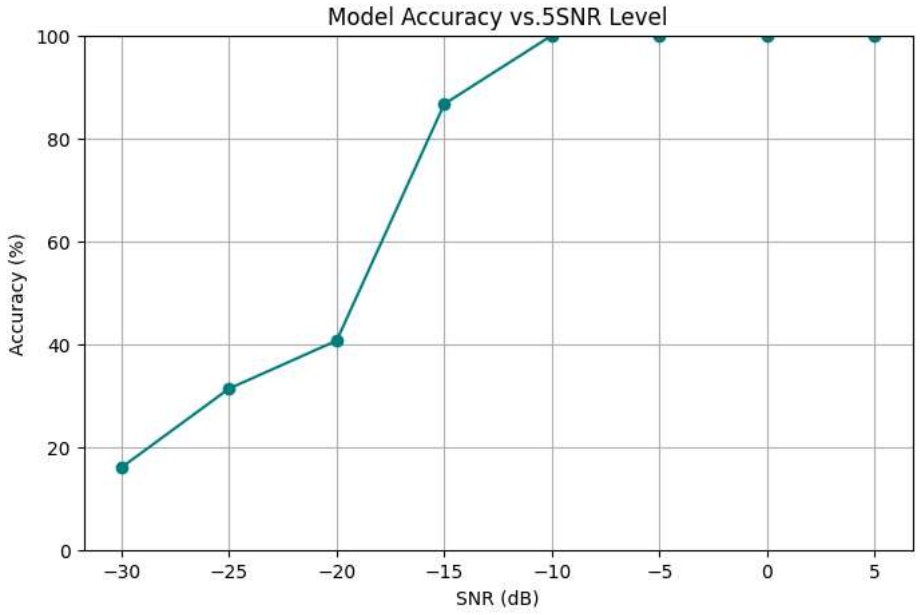
```
Epoch 1/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 51s 653ms/step - accuracy: 0.2654 - loss: 12.7904 - val_accuracy: 0.4142 - val_loss: 1.4955
Epoch 2/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 0.4541 - loss: 1.3876 - val_accuracy: 0.6442 - val_loss: 0.9171
Epoch 3/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 0.7064 - loss: 0.7644 - val_accuracy: 0.7033 - val_loss: 0.7307
Epoch 4/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 0.8971 - loss: 0.3913 - val_accuracy: 0.6938 - val_loss: 0.8033
Epoch 5/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 0.9869 - loss: 0.1422 - val_accuracy: 0.7175 - val_loss: 0.7596
Epoch 6/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 0.0344 - val_accuracy: 0.7163 - val_loss: 0.8637
Epoch 7/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 0.0104 - val_accuracy: 0.7167 - val_loss: 0.8816
Epoch 8/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 0.0052 - val_accuracy: 0.7171 - val_loss: 0.8980
Epoch 9/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 0.7183 - val_loss: 0.9180
Epoch 10/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 0.7179 - val_loss: 0.9657
Epoch 11/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.7175 - val_loss: 0.9665
Epoch 12/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 85ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.7196 - val_loss: 0.9750
Epoch 13/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.7183 - val_loss: 1.0002
Epoch 14/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 9.5341e-04 - val_accuracy: 0.7158 - val_loss: 1.0053
Epoch 15/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - accuracy: 1.0000 - loss: 7.9797e-04 - val_accuracy: 0.7183 - val_loss: 1.0123
Epoch 16/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 1.0000 - loss: 6.9256e-04 - val_accuracy: 0.7138 - val_loss: 1.0335
Epoch 17/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 5.8565e-04 - val_accuracy: 0.7175 - val_loss: 1.0317
Epoch 18/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 1.0000 - loss: 5.1334e-04 - val_accuracy: 0.7171 - val_loss: 1.0376
Epoch 19/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 1.0000 - loss: 4.3578e-04 - val_accuracy: 0.7163 - val_loss: 1.0525
Epoch 20/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 4s 81ms/step - accuracy: 1.0000 - loss: 3.9853e-04 - val_accuracy: 0.7183 - val_loss: 1.0635
<keras.src.callbacks.history.History at 0x7d6b599fb6d0>
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\n🔍 Final Test Accuracy: {test_acc * 100:.2f}%")
```

```
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.4431 - loss: 2.0831

🔍 Final Test Accuracy: 71.83%
```

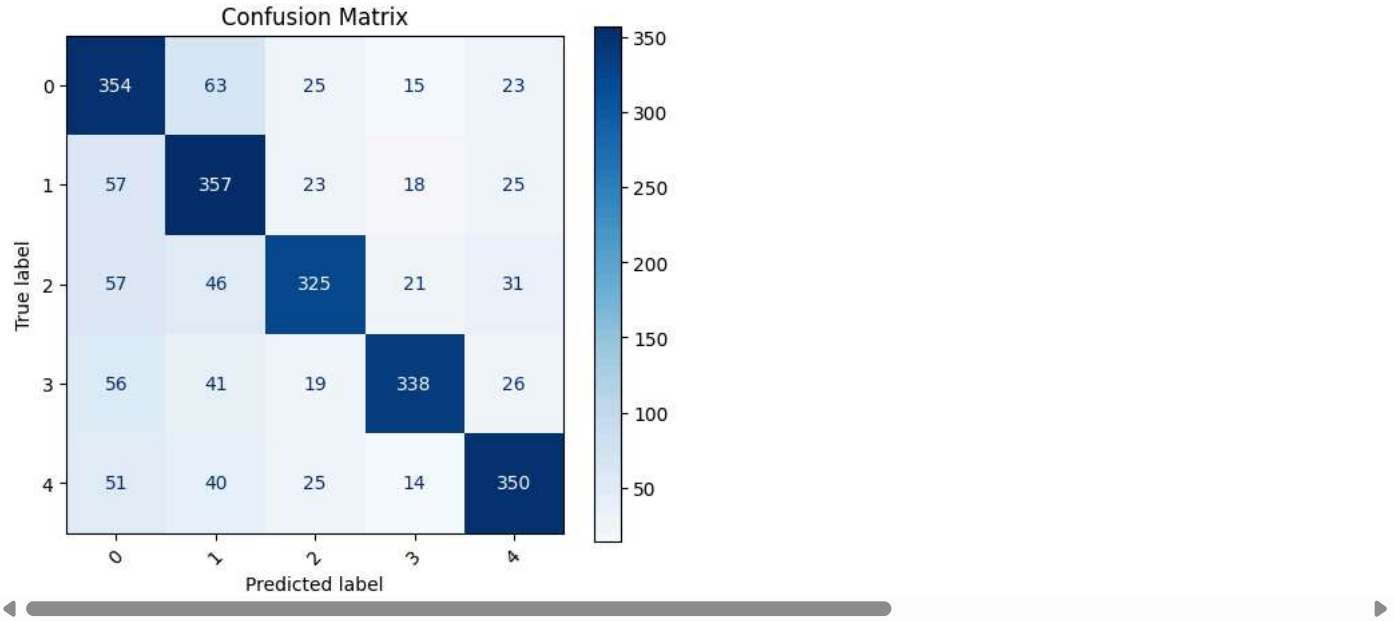
```
# SNR-wise analysis (per mentor's instructions)
import matplotlib.pyplot as plt

print("Encoded classes:", encoder.categories_)
print("One-hot sample:", y_train[:5])
print("Reversed (decoded) sample:", encoder.inverse_transform(y_train[:5]))

# Step 1: Define SNR levels and how test data is structured
snr_levels = [-30, -25, -20, -15, -10, -5, 0, 5]
samples_per_snr = 60 * 5  # 60 samples per class × 5 classes = 300 per SNR level

## ========== PLOT SNR-WISE ACCURACY (USING INVERSE TRANSFORM) ==========

def accuracy_per_snr(model, X_test, y_test, snr_levels, samples_per_snr, encoder):
    snr_acc = {}
    for i, snr in enumerate(snr_levels):
        start = i * samples_per_snr
        end = (i + 1) * samples_per_snr

        X_snr = X_test[start:end]
        y_snr = y_test[start:end]

        y_pred = model.predict(X_snr)

        # Use inverse_transform to decode one-hot back to labels
        y_true_labels = encoder.inverse_transform(y_snr).flatten()
        y_pred_labels = encoder.inverse_transform(y_pred).flatten()

        acc = np.mean(y_pred_labels == y_true_labels)
        snr_acc[snr] = acc
        print(f"SNR {snr} dB: Accuracy = {acc * 100:.2f}%")
    return snr_acc

def plot_snr_accuracy(snr_acc):
    plt.figure(figsize=(8, 5))
    plt.plot(list(snr_acc.keys()), [v * 100 for v in snr_acc.values()],
             marker='o', linestyle='-', color='teal')
    plt.xlabel("SNR (dB)")
    plt.ylabel("Accuracy (%)")
    plt.title("Model Accuracy vs.5SNR Level")
    plt.xticks(list(snr_acc.keys()))
    plt.ylim(0, 100)
    plt.grid(True)
    plt.show()

# Example usage (ensure `snr_levels` and `samples_per_snr` are defined)
snr_acc = accuracy_per_snr(model, X_test, y_test, snr_levels, samples_per_snr, encoder)
plot_snr_accuracy(snr_acc)
```

```
Encoded classes: [array([0., 1., 2., 3., 4.])]
One-hot sample: [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]]
Reversed (decoded) sample: [[0.]
 [1.]
 [1.]
 [3.]
 [2.]]
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
SNR -30 dB: Accuracy = 16.00%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
SNR -25 dB: Accuracy = 31.33%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
SNR -20 dB: Accuracy = 40.67%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
SNR -15 dB: Accuracy = 86.67%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
SNR -10 dB: Accuracy = 100.00%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
SNR -5 dB: Accuracy = 100.00%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
SNR 0 dB: Accuracy = 100.00%
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
SNR 5 dB: Accuracy = 100.00%
```




```
# ========== PLOT CONFUSION MATRIX ==========

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_conf_matrix(model, X_test, y_test, class_names=None):
    y_pred = model.predict(X_test).argmax(axis=1)
    y_true = y_test.argmax(axis=1)
    cm = confusion_matrix(y_true, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
    fig, ax = plt.subplots(figsize=(6, 5))
    disp.plot(ax=ax, cmap='Blues', xticks_rotation=45)
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()

# Optional: Pass class names
plot_conf_matrix(model, X_test, y_test, class_names=[0, 1, 2, 3, 4])
```

```
75/75 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```



Code for Preprocessing Jet-Colored Images

```
# Code for Preprocessing Jet-Colored Images
import matplotlib.pyplot as plt
from PIL import Image
import io

# Convert a normalized spectrogram into jet-colored RGB image
def signature_to_rgb_image(sig_1d, height=256, width=922, size=(224, 224)):
    sig_2d = sig_1d.reshape(height, width)

    fig, ax = plt.subplots(figsize=(2.56, 0.922), dpi=100)
    ax.imshow(sig_2d, cmap='jet', aspect='auto')
    ax.axis('off')

    buf = io.BytesIO()
    plt.savefig(buf, format='png', bbox_inches='tight', pad_inches=0)
    plt.close(fig)

    buf.seek(0)
    img = Image.open(buf).convert('RGB')  # Ensure RGB format
    img_resized = img.resize(size)
    img_array = np.asarray(img_resized).astype(np.float32)

    # Normalize to [-1, 1]
    img_normalized = ((img_array / 255.0) - 0.5) * 2
    return img_normalized


# One-Hot Encode Labels

from sklearn.preprocessing import OneHotEncoder

# ========== ONE-HOT ENCODING WITH SAFETY CHECK ==========
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
y_train = encoder.fit_transform(y_train_1.reshape(-1, 1))
y_test = encoder.transform(y_test_1.reshape(-1, 1))


#  Convert Entire Dataset to RGB Jet Format
print("Converting training images...")
X_train_rgb = np.array([signature_to_rgb_image(sig) for sig in X_train_1])

print("Converting test images...")
X_test_rgb = np.array([signature_to_rgb_image(sig) for sig in X_test_1])

print("Done! Shapes:")
print("X_train_rgb:", X_train_rgb.shape)
print("X_test_rgb :", X_test_rgb.shape)
```

```
Converting training images...
Converting test images...
Done! Shapes:
X_train_rgb: (5600, 224, 224, 3)
X_test_rgb : (2400, 224, 224, 3)
```

```
# Now let us work with more complex model to improve even more

import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, InputLayer
from tensorflow.keras.optimizers import Adam

# Build the correct model
model = Sequential()

# 1st Convolutional Layer
model.add(InputLayer(input_shape=(224, 224, 3)))  # updated to RGB image size
model.add(Conv2D(6, kernel_size=(5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 2nd Convolutional Layer
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Flatten & Fully Connected Layers
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))

# Output Layer for 5 UAV classes
model.add(Dense(5, activation='softmax'))

model.compile(loss=keras.losses.CategoricalCrossentropy(), optimizer=Adam(), metrics=['accuracy'])

# Train the model
model.fit(X_train_rgb, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_test_rgb, y_test))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27: UserWarning: Argument `input_shape` is deprecated. Use
  warnings.warn(
Epoch 1/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 25s 313ms/step - accuracy: 0.2674 - loss: 3.1316 - val_accuracy: 0.5263 - val_loss: 1.1536
Epoch 2/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.5856 - loss: 1.0601 - val_accuracy: 0.6258 - val_loss: 0.9158
Epoch 3/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.6753 - loss: 0.8327 - val_accuracy: 0.6421 - val_loss: 0.9040
Epoch 4/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.7089 - loss: 0.7591 - val_accuracy: 0.6550 - val_loss: 0.8441
Epoch 5/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.7315 - loss: 0.6871 - val_accuracy: 0.6612 - val_loss: 0.8458
Epoch 6/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.7853 - loss: 0.5954 - val_accuracy: 0.6700 - val_loss: 0.8140
Epoch 7/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.8353 - loss: 0.4839 - val_accuracy: 0.6671 - val_loss: 0.8448
Epoch 8/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.8718 - loss: 0.4078 - val_accuracy: 0.6612 - val_loss: 0.9247
Epoch 9/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9135 - loss: 0.2921 - val_accuracy: 0.6667 - val_loss: 0.9219
Epoch 10/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9573 - loss: 0.2014 - val_accuracy: 0.6700 - val_loss: 0.9813
Epoch 11/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9769 - loss: 0.1254 - val_accuracy: 0.6646 - val_loss: 1.1096
Epoch 12/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9937 - loss: 0.0734 - val_accuracy: 0.6642 - val_loss: 1.2048
Epoch 13/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9983 - loss: 0.0449 - val_accuracy: 0.6650 - val_loss: 1.3650
Epoch 14/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9979 - loss: 0.0269 - val_accuracy: 0.6612 - val_loss: 1.4646
Epoch 15/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 1.0000 - loss: 0.0131 - val_accuracy: 0.6679 - val_loss: 1.6000
Epoch 16/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 1.0000 - loss: 0.0078 - val_accuracy: 0.6662 - val_loss: 1.6567
Epoch 17/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 1.0000 - loss: 0.0048 - val_accuracy: 0.6662 - val_loss: 1.7451
Epoch 18/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.6654 - val_loss: 1.8226
Epoch 19/20
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.6629 - val_loss: 1.8777
```