In [1]:
```
from tensorflow.keras import backend as K
import gc
K.clear_session()
gc.collect()
```

```
WARNING:tensorflow:From C:\Users\NIDHI KAPADIYA\anaconda3\envs\py39\lib\site-packages\keras\src\backend\common\global_state.py:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.
```

Out[1]: 0

Sounds of Seattle Birds

- Goal is to predict bird species based on sound using Neural Network.
- There are three parts:

1. Binary classification on any 2 species.
2. Multiclass classification on 12 species.
3. External data test on best model obtained form multiclass training.

Starting with data exploration

EDA

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df = pd.read_csv("E:\\Spring Quarter 2025\\SML2\\Homework3\\bird_metadata.csv")
df.head(5)
```

Out[3]:

	Unnamed: 0	rating	playback_used	ebird_code	channels	date	duration	filename	species	title	...	xc_id
0	166	5.0	no	amecro	2 (stereo)	2020-07-12	44	XC575220.mp3	American Crow	XC575220 American Crow (Corvus brachyrhynchos)	...	575220
1	169	5.0	no	amecro	2 (stereo)	2020-06-17	41	XC569711.mp3	American Crow	XC569711 American Crow (Corvus brachyrhynchos ...	...	569711
2	171	5.0	no	amecro	1 (mono)	2020-06-14	78	XC568365.mp3	American Crow	XC568365 American Crow (Corvus brachyrhynchos ...	...	568365
3	173	5.0	no	amecro	2 (stereo)	0000-00-00	52	XC561291.mp3	American Crow	XC561291 American Crow (Corvus brachyrhynchos)	...	561291
4	174	5.0	no	amecro	1 (mono)	2019-04-18	13	XC556240.mp3	American Crow	XC556240 American Crow (Corvus brachyrhynchos)	...	556240

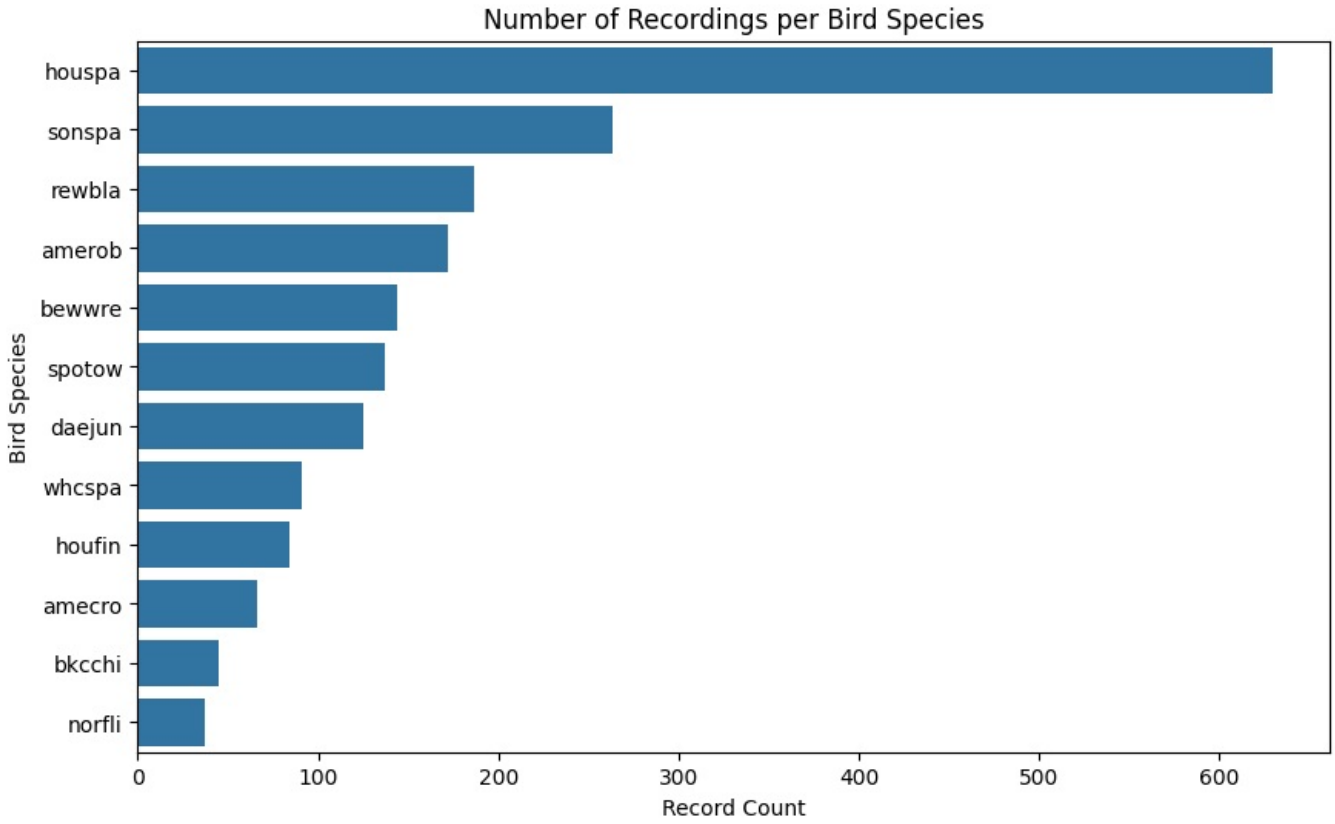
5 rows × 31 columns

In [4]:
```
df.describe()
```

Out[4]:

	Unnamed: 0	rating	duration	latitude	background	xc_id	longitude
count	1981.000000	1981.000000	1981.000000	1956.000000	0.0	1981.000000	1956.000000
mean	12029.902070	3.858152	51.772842	38.171789	NaN	361785.941949	-71.628008
std	7054.574167	0.680590	72.745494	13.832487	NaN	162729.163296	54.181736
min	166.000000	3.000000	3.000000	-41.204000	NaN	11446.000000	-165.405300
25%	7042.000000	3.000000	15.000000	33.901850	NaN	199496.000000	-114.157275
50%	11493.000000	4.000000	31.000000	39.654900	NaN	364400.000000	-88.113450
75%	19033.000000	4.000000	60.000000	45.876750	NaN	533616.000000	-7.630300
max	22783.000000	5.000000	846.000000	69.577800	NaN	588021.000000	153.139800

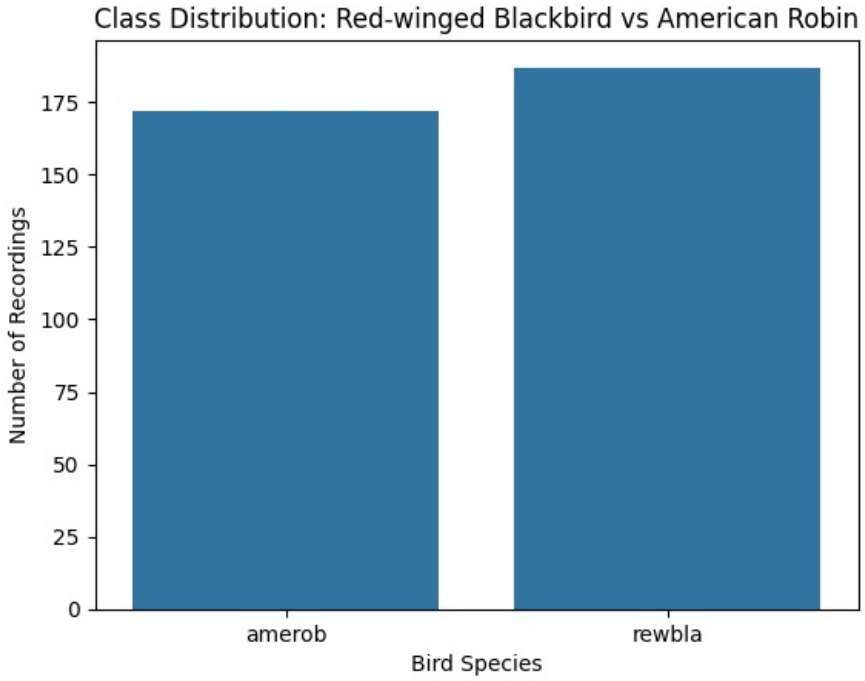
In [5]:
```
# Samples per class
plt.figure(figsize=(10,6))
sns.countplot(y='ebird_code', data=df, order=df['ebird_code'].value_counts().index)
plt.title('Number of Recordings per Bird Species')
plt.xlabel('Record Count')
plt.ylabel('Bird Species')
plt.show()
```



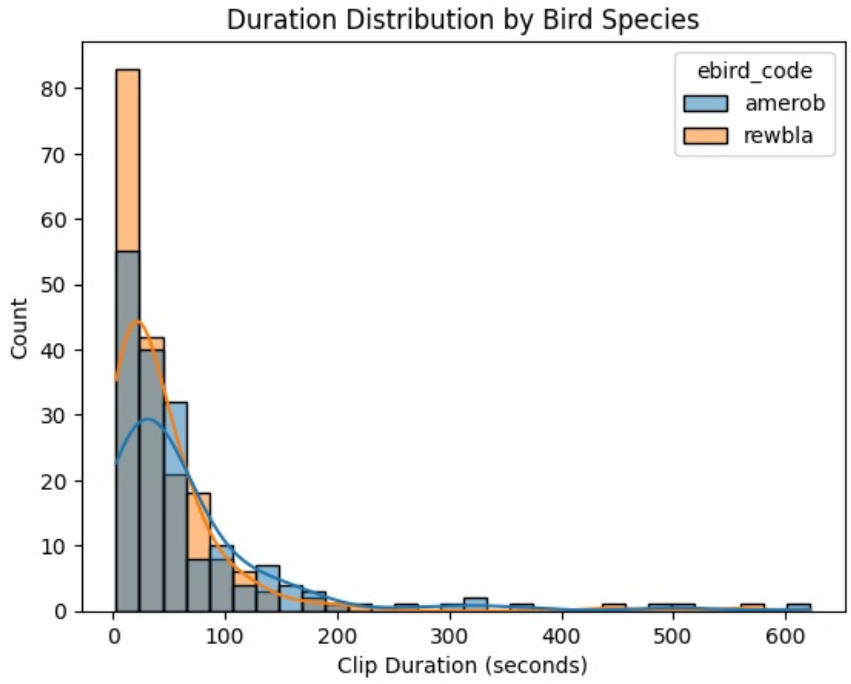
- From the plot, majority samples are of house sparrow. While, Northen flicker and Black-capped chikadae is in minority class.
- We will perform binary classification on Red-winged black bird and American Robin as they have around same samples.

In [8]:
```
binary_df = df[df['ebird_code'].isin(['rewbla', 'amerob'])].copy()
```

In [9]:
```
sns.countplot(data=binary_df, x='ebird_code')
plt.title('Class Distribution: Red-winged Blackbird vs American Robin')
plt.xlabel('Bird Species')
plt.ylabel('Number of Recordings')
plt.show()
```



In [10]:
```
sns.histplot(data=binary_df, x='duration', hue='ebird_code', bins=30, kde=True)
plt.title('Duration Distribution by Bird Species')
plt.xlabel('Clip Duration (seconds)')
plt.ylabel('Count')
plt.show()
```



Binary Classification - CNN

Code flow:

1. Data Preparation
2. Spliting data on 70-30
3. Building basic 3 CNN models
4. Evaluating and finding 2 best out of 3CNN models
5. Trying different batch sizes and dropout on thos 2 models
6. Discusssing Result
7. Finding best model

Red-winged Blackbird vs American Robin

Step1: Load Dependencies

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import tensorflow as tf
from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

Step2 : Data Loading

In [12]:
```
bird_sound = h5py.File("E:\\Spring Quarter 2025\\SML2\\Homework3\\bird_spectrograms.hdf5", 'r')
print(list(bird_sound.keys()))
```

```
['amecro', 'amerob', 'bewwre', 'bkcchi', 'daejun', 'houfin', 'houspa', 'norfli', 'rewbla', 'sonspa', 'spotow', '
whcspa']
```

In [13]:
```
bird1 = bird_sound['amerob'][:]
bird2 = bird_sound['rewbla'][:]

print(bird1.shape)
print(bird2.shape)
```

```
(128, 517, 172)
(128, 517, 187)
```

Step3: Preaparing Data

In [14]:
```
# Transforming data and assinging labels 0 to American robin and 1 to Red-winged Blackbird

X_bird1= np.transpose(bird1,(2, 0, 1))
X_bird2 = np.transpose(bird2, (2, 0, 1))
y_bird1 = np.zeros(172)
y_bird2 = np.ones(187)
```

In [15]:
```
# Concaenating both species data
X = np.concatenate([X_bird1, X_bird2],axis=0)
y = np.concatenate([y_bird1,y_bird2],axis=0)
```

In [16]:
```
X = X.reshape((359, 128, 517, 1))
print(X.shape)
```

```
(359, 128, 517, 1)
```

Step4: Train-Test Split

In [17]:
```
# Splitting data into 70-30 for training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y)
```

Step5 : Building Model

In [18]:
```
# First binary CNN model
model1_binary = Sequential()

# Step 1 - Convolution
model1_binary.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))

# Step 2 - Pooling
model1_binary.add(MaxPooling2D(pool_size=(2, 2)))

# Adding a second convolutional layer
model1_binary.add(Conv2D(32, (3, 3), activation='relu'))

# Step 3 - Flattening
model1_binary.add(Flatten())

# Step 4 - Full connection
model1_binary.add(Dense(units=64, activation='relu'))
model1_binary.add(Dropout(0.3))
model1_binary.add(Dense(units=1, activation='sigmoid'))
```

```
C:\Users\NIDHI KAPADIYA\anaconda3\envs\py39\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer u
sing an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Step6: Training Model

In [19]:
```
# Metrics are accuracy,precision,recall and auc
model1_binary.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ]
)


import time

start_time = time.time()

history1 = model1_binary.fit(
    X_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(X_test, y_test)
)
end_time = time.time()
time1 = end_time - start_time
print(f"Training time: {time1:.2f} seconds")
```

```
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.4915 - auc: 0.4980 - loss: 707.5485 - precision: 0.4898 - reca
ll: 0.6092 - val_accuracy: 0.4815 - val_auc: 0.5000 - val_loss: 252.3771 - val_precision: 0.0000e+00 - val_recal
l: 0.0000e+00
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.4521 - auc: 0.4577 - loss: 220.8301 - precision: 0.0000e+00 -
recall: 0.0000e+00 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 53.1764 - val_precision: 0.5185 - val_re
call: 1.0000
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 741ms/step - accuracy: 0.4760 - auc: 0.4165 - loss: 35.9204 - precision: 0.4752 - re
call: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 19.0970 - val_precision: 0.5185 - val_recall:
1.0000
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 758ms/step - accuracy: 0.5088 - auc: 0.4878 - loss: 23.9718 - precision: 0.5132 - re
call: 0.6240 - val_accuracy: 0.5000 - val_auc: 0.4935 - val_loss: 0.7070 - val_precision: 0.5114 - val_recall: 0
.8036
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 778ms/step - accuracy: 0.5726 - auc: 0.5712 - loss: 4.1283 - precision: 0.5453 - rec
all: 0.8459 - val_accuracy: 0.4815 - val_auc: 0.5000 - val_loss: 10.3164 - val_precision: 0.0000e+00 - val_recal
l: 0.0000e+00
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 940ms/step - accuracy: 0.5532 - auc: 0.5359 - loss: 8.9385 - precision: 0.5595 - rec
all: 0.7281 - val_accuracy: 0.5000 - val_auc: 0.4811 - val_loss: 0.6941 - val_precision: 0.5094 - val_recall: 0.
9643
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 963ms/step - accuracy: 0.5139 - auc: 0.4839 - loss: 1.2628 - precision: 0.5156 - rec
all: 0.9939 - val_accuracy: 0.4907 - val_auc: 0.4818 - val_loss: 0.6947 - val_precision: 0.5048 - val_recall: 0.
9464
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 755ms/step - accuracy: 0.4531 - auc: 0.4597 - loss: 1.1508 - precision: 0.4700 - rec
all: 0.6710 - val_accuracy: 0.5185 - val_auc: 0.5936 - val_loss: 0.7438 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 689ms/step - accuracy: 0.5269 - auc: 0.5078 - loss: 1.0132 - precision: 0.5341 - rec
all: 0.7791 - val_accuracy: 0.5185 - val_auc: 0.6401 - val_loss: 0.6897 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.5567 - auc: 0.6082 - loss: 0.6740 - precision: 0.5560 - recall
: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.6679 - val_loss: 0.6837 - val_precision: 0.5185 - val_recall: 1.000
0
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.5304 - auc: 0.7339 - loss: 0.6731 - precision: 0.5201 - recall
: 1.0000 - val_accuracy: 0.5278 - val_auc: 0.7419 - val_loss: 0.6790 - val_precision: 0.5234 - val_recall: 1.000
0
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.6568 - auc: 0.8200 - loss: 0.6425 - precision: 0.6125 - recall
: 0.9847 - val_accuracy: 0.5370 - val_auc: 0.7728 - val_loss: 0.8400 - val_precision: 1.0000 - val_recall: 0.107
1
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.6748 - auc: 0.7817 - loss: 0.6203 - precision: 0.7331 - recall
: 0.5940 - val_accuracy: 0.6667 - val_auc: 0.7009 - val_loss: 0.6522 - val_precision: 0.6351 - val_recall: 0.839
3
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.8253 - auc: 0.9065 - loss: 0.3882 - precision: 0.7935 - recall
: 0.9171 - val_accuracy: 0.6852 - val_auc: 0.7572 - val_loss: 0.5846 - val_precision: 0.6964 - val_recall: 0.696
4
```

```
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.9108 - auc: 0.9815 - loss: 0.2305 - precision: 0.8986 - recall
: 0.9277 - val_accuracy: 0.6574 - val_auc: 0.7758 - val_loss: 0.7116 - val_precision: 0.6267 - val_recall: 0.839
3
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.9423 - auc: 0.9864 - loss: 0.1609 - precision: 0.9390 - recall
: 0.9545 - val_accuracy: 0.6759 - val_auc: 0.7727 - val_loss: 0.7258 - val_precision: 0.7234 - val_recall: 0.607
1
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9533 - auc: 0.9938 - loss: 0.1262 - precision: 0.9625 - recall
: 0.9483 - val_accuracy: 0.7222 - val_auc: 0.7828 - val_loss: 0.7035 - val_precision: 0.7826 - val_recall: 0.642
9
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9843 - auc: 0.9965 - loss: 0.0914 - precision: 0.9820 - recall
: 0.9892 - val_accuracy: 0.6759 - val_auc: 0.7728 - val_loss: 0.8627 - val_precision: 0.6981 - val_recall: 0.660
7
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.9858 - auc: 0.9996 - loss: 0.0649 - precision: 1.0000 - recall
: 0.9731 - val_accuracy: 0.6759 - val_auc: 0.7684 - val_loss: 1.0682 - val_precision: 0.7143 - val_recall: 0.625
0
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9769 - auc: 0.9990 - loss: 0.0591 - precision: 0.9844 - recall
: 0.9736 - val_accuracy: 0.6574 - val_auc: 0.7512 - val_loss: 1.1479 - val_precision: 0.6377 - val_recall: 0.785
7
Epoch 21/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9889 - auc: 0.9997 - loss: 0.0307 - precision: 0.9868 - recall
: 0.9918 - val_accuracy: 0.6481 - val_auc: 0.7474 - val_loss: 1.2609 - val_precision: 0.6364 - val_recall: 0.750
0
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9707 - auc: 0.9968 - loss: 0.0852 - precision: 0.9426 - recall
: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.7883 - val_loss: 0.9047 - val_precision: 0.7391 - val_recall: 0.607
1
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9911 - auc: 0.9997 - loss: 0.0344 - precision: 0.9860 - recall
: 0.9969 - val_accuracy: 0.7037 - val_auc: 0.7809 - val_loss: 1.1868 - val_precision: 0.8750 - val_recall: 0.500
0
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9947 - auc: 0.9998 - loss: 0.0244 - precision: 0.9900 - recall
: 1.0000 - val_accuracy: 0.7037 - val_auc: 0.7993 - val_loss: 1.0625 - val_precision: 0.8158 - val_recall: 0.553
6
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 987ms/step - accuracy: 0.9890 - auc: 0.9997 - loss: 0.0331 - precision: 0.9918 - rec
all: 0.9869 - val_accuracy: 0.6852 - val_auc: 0.7946 - val_loss: 1.0095 - val_precision: 0.7037 - val_recall: 0.
6786
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9832 - auc: 0.9993 - loss: 0.0416 - precision: 0.9784 - recall
: 0.9899 - val_accuracy: 0.6944 - val_auc: 0.8001 - val_loss: 1.1281 - val_precision: 0.7170 - val_recall: 0.678
6
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9984 - auc: 0.9999 - loss: 0.0120 - precision: 1.0000 - recall
: 0.9969 - val_accuracy: 0.6944 - val_auc: 0.8046 - val_loss: 1.1186 - val_precision: 0.7347 - val_recall: 0.642
9
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 980ms/step - accuracy: 0.9843 - auc: 0.9999 - loss: 0.0245 - precision: 0.9765 - rec
all: 0.9951 - val_accuracy: 0.6759 - val_auc: 0.7746 - val_loss: 1.2735 - val_precision: 0.6981 - val_recall: 0.
6607
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 961ms/step - accuracy: 0.9916 - auc: 0.9998 - loss: 0.0275 - precision: 0.9969 - rec
all: 0.9873 - val_accuracy: 0.6481 - val_auc: 0.7785 - val_loss: 1.1998 - val_precision: 0.6324 - val_recall: 0.
7679
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9974 - auc: 1.0000 - loss: 0.0177 - precision: 0.9950 - recall
: 1.0000 - val_accuracy: 0.6574 - val_auc: 0.7776 - val_loss: 1.2803 - val_precision: 0.6338 - val_recall: 0.803
6
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9942 - auc: 1.0000 - loss: 0.0172 - precision: 0.9893 - recall
: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7593 - val_loss: 1.6135 - val_precision: 0.6351 - val_recall: 0.839
3
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 987ms/step - accuracy: 0.9916 - auc: 0.9996 - loss: 0.0223 - precision: 0.9917 - rec
all: 0.9918 - val_accuracy: 0.6852 - val_auc: 0.7701 - val_loss: 1.4269 - val_precision: 0.6618 - val_recall: 0.
8036
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0120 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7644 - val_loss: 1.3588 - val_precision: 0.6471 - val_recall: 0.785
7
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9854 - auc: 0.9997 - loss: 0.0183 - precision: 0.9863 - recall
: 0.9863 - val_accuracy: 0.6574 - val_auc: 0.7497 - val_loss: 1.4626 - val_precision: 0.6418 - val_recall: 0.767
9
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 936ms/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0176 - precision: 1.0000 - rec
all: 1.0000 - val_accuracy: 0.6944 - val_auc: 0.7663 - val_loss: 1.4776 - val_precision: 0.7674 - val_recall: 0.
```

```
5893
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 745ms/step - accuracy: 0.9916 - auc: 1.0000 - loss: 0.0127 - precision: 1.0000 - rec
all: 0.9840 - val_accuracy: 0.7037 - val_auc: 0.7730 - val_loss: 1.4211 - val_precision: 0.7727 - val_recall: 0.
6071
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 880ms/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0169 - precision: 1.0000 - rec
all: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7763 - val_loss: 1.3222 - val_precision: 0.6981 - val_recall: 0.
6607
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9916 - auc: 0.9999 - loss: 0.0153 - precision: 0.9892 - recall
: 0.9951 - val_accuracy: 0.6759 - val_auc: 0.7728 - val_loss: 1.3560 - val_precision: 0.6780 - val_recall: 0.714
3
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.9958 - auc: 1.0000 - loss: 0.0123 - precision: 0.9921 - recall
: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7837 - val_loss: 1.2445 - val_precision: 0.6842 - val_recall: 0.696
4
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 939ms/step - accuracy: 0.9958 - auc: 0.9999 - loss: 0.0166 - precision: 0.9920 - rec
all: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7763 - val_loss: 1.3156 - val_precision: 0.6786 - val_recall: 0.
6786
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0080 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.7037 - val_auc: 0.7708 - val_loss: 1.5380 - val_precision: 0.7500 - val_recall: 0.642
9
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9927 - auc: 1.0000 - loss: 0.0122 - precision: 0.9872 - recall
: 1.0000 - val_accuracy: 0.6481 - val_auc: 0.7395 - val_loss: 2.0225 - val_precision: 0.7368 - val_recall: 0.500
0
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 995ms/step - accuracy: 0.9958 - auc: 1.0000 - loss: 0.0125 - precision: 1.0000 - rec
all: 0.9920 - val_accuracy: 0.6852 - val_auc: 0.7723 - val_loss: 1.5241 - val_precision: 0.6964 - val_recall: 0.
6964
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0042 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6944 - val_auc: 0.7751 - val_loss: 1.4877 - val_precision: 0.6885 - val_recall: 0.750
0
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.9974 - auc: 1.0000 - loss: 0.0110 - precision: 0.9949 - recall
: 1.0000 - val_accuracy: 0.7037 - val_auc: 0.7787 - val_loss: 1.4653 - val_precision: 0.6935 - val_recall: 0.767
9
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 958ms/step - accuracy: 0.9858 - auc: 0.9997 - loss: 0.0279 - precision: 0.9874 - rec
all: 0.9867 - val_accuracy: 0.7037 - val_auc: 0.7879 - val_loss: 1.3975 - val_precision: 0.6875 - val_recall: 0.
7857
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 954ms/step - accuracy: 0.9958 - auc: 1.0000 - loss: 0.0126 - precision: 0.9920 - rec
all: 1.0000 - val_accuracy: 0.7130 - val_auc: 0.7905 - val_loss: 1.3852 - val_precision: 0.7049 - val_recall: 0.
7679
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.9958 - auc: 1.0000 - loss: 0.0093 - precision: 1.0000 - recall
: 0.9920 - val_accuracy: 0.7130 - val_auc: 0.7821 - val_loss: 1.4631 - val_precision: 0.7049 - val_recall: 0.767
9
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 977ms/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0047 - precision: 1.0000 - rec
all: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7694 - val_loss: 1.6296 - val_precision: 0.6615 - val_recall: 0.
7679
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 895ms/step - accuracy: 0.9932 - auc: 0.9999 - loss: 0.0186 - precision: 0.9950 - rec
all: 0.9921 - val_accuracy: 0.7037 - val_auc: 0.7618 - val_loss: 1.7069 - val_precision: 0.6765 - val_recall: 0.
8214
Training time: 226.85 seconds
```

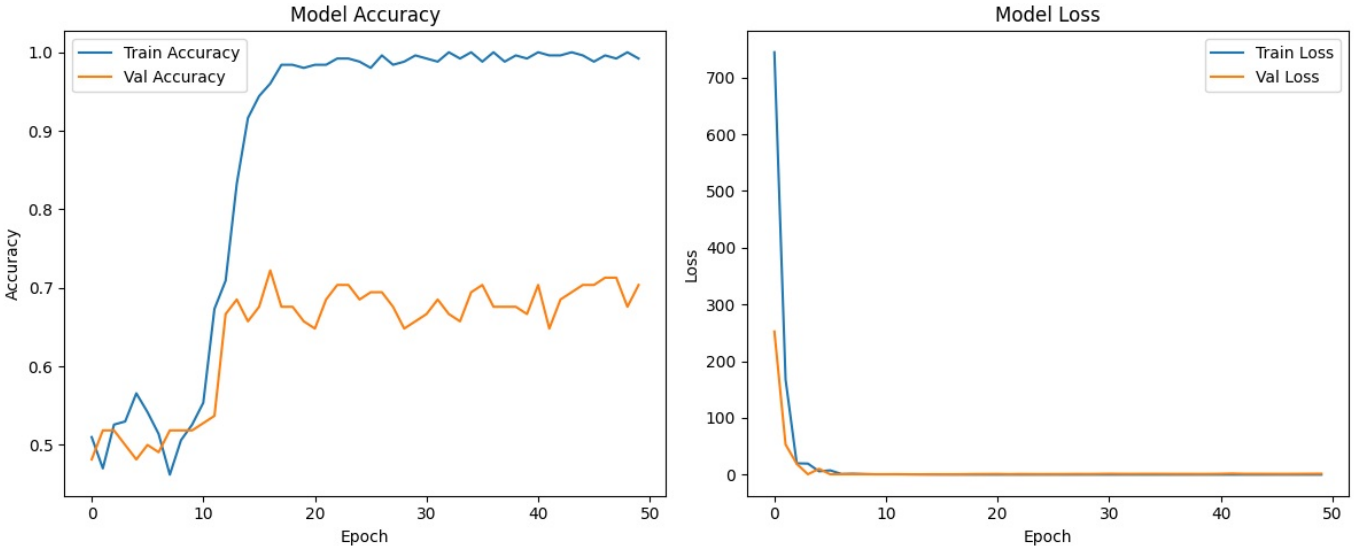
In [20]:
```
model1_binary.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 515, 16)	160
max_pooling2d (MaxPooling2D)	(None, 63, 257, 16)	0
conv2d_1 (Conv2D)	(None, 61, 255, 32)	4,640
flatten (Flatten)	(None, 497760)	0
dense (Dense)	(None, 64)	31,856,704
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 95,584,709 (364.63 MB)
Trainable params: 31,861,569 (121.54 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 63,723,140 (243.08 MB)

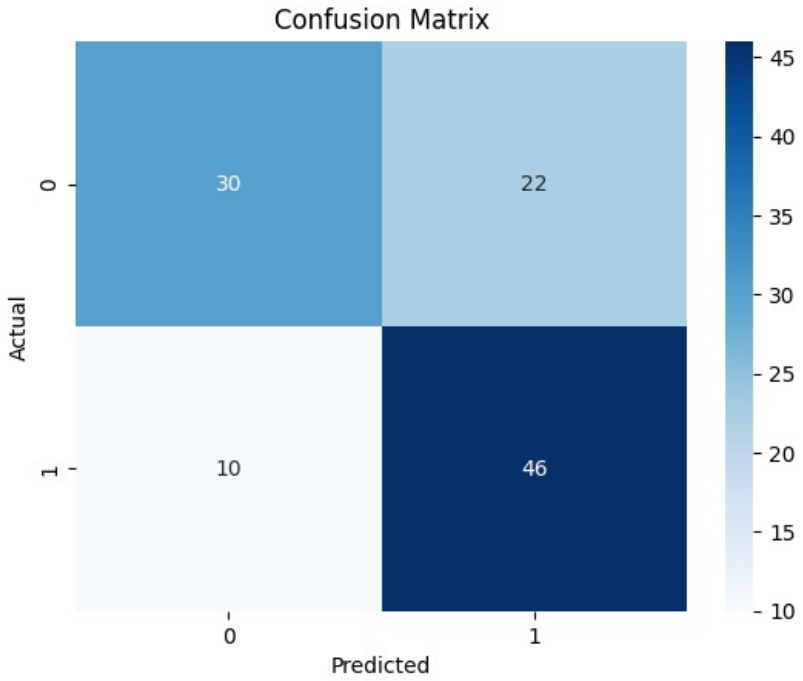
In [21]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history1.history['accuracy'], label='Train Accuracy')
plt.plot(history1.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history1.history['loss'], label='Train Loss')
plt.plot(history1.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



- After around 15 epochs, the model's training accuracy keeps improving, but validation accuracy stays around 70%.
- This means the model starts overfitting after 15 epochs, and more training does not help improve validation performance.

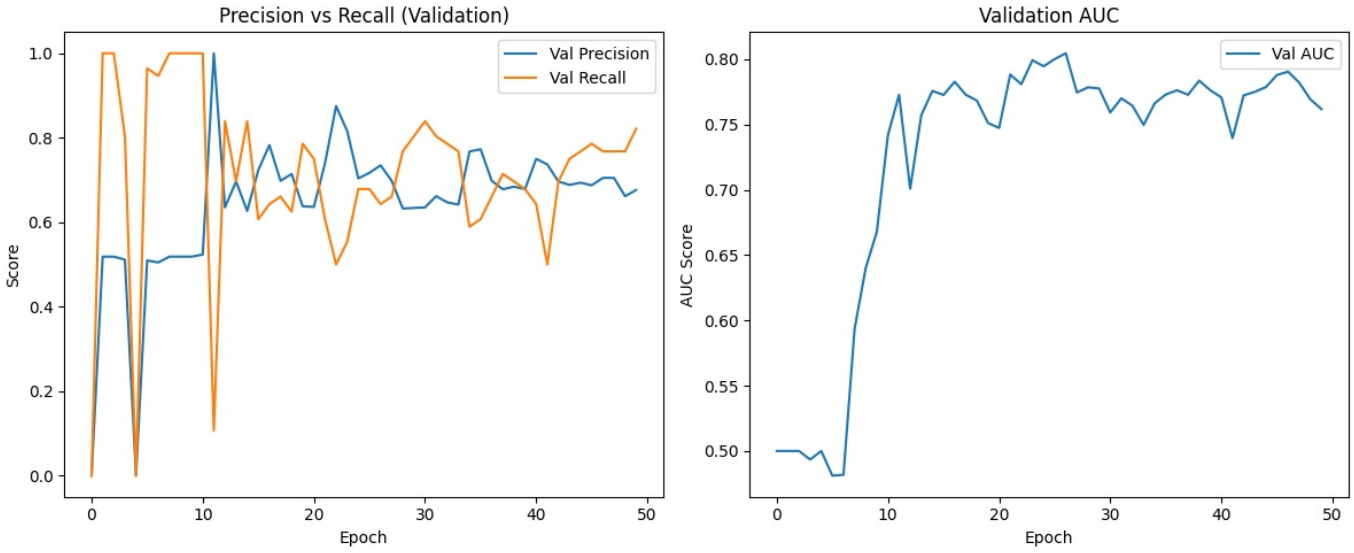
In [22]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

loss1, accuracy1, precision1, recall1, auc1 = model1_binary.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy1:.4f}")
print(f"Test Precision: {precision1:.4f}")
print(f"Test Recall: {recall1:.4f}")
print(f"Test AUC: {auc1:.4f}")

y_pred_probs = model1_binary.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype("int32").flatten()
y_true = y_test.flatten()

cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 170ms/step - accuracy: 0.7200 - auc: 0.7796 - loss: 1.9087 - precision: 0.6886 - rec
all: 0.8099
Test Accuracy: 0.7037
Test Precision: 0.6765
Test Recall: 0.8214
Test AUC: 0.7618
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 210ms/step
```



- The heatmap shows that the model correctly predicted most Red-winged Blackbird calls (46).
- However, some American Robin samples were misclassified as Red-winged Blackbird (22).

In [23]:
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history1.history['val_precision'], label='Val Precision')
plt.plot(history1.history['val_recall'], label='Val Recall')
plt.title('Precision vs Recall (Validation)')
plt.xlabel('Epoch')
plt.ylabel('Score')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history1.history['val_auc'], label='Val AUC')
plt.title('Validation AUC')
plt.xlabel('Epoch')
plt.ylabel('AUC Score')
plt.legend()

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



- The precision and recall scores become more stable after around epoch 15, showing improved prediction consistency.
- The AUC score increases and stays above 0.75,indicates that the model is good in prediction.

CNN-Model2

In [24]:
```
# Defining model
model2_binary = Sequential()
model2_binary.add(Conv2D(8, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model2_binary.add(MaxPooling2D(pool_size=(2, 2)))

model2_binary.add(Conv2D(16, (3, 3), activation='relu'))
model2_binary.add(MaxPooling2D(pool_size=(2, 2)))

model2_binary.add(Flatten())
model2_binary.add(Dense(32, activation='relu'))
model2_binary.add(Dropout(0.3))
model2_binary.add(Dense(1, activation='sigmoid'))
```

```
C:\Users\NIDHI KAPADIYA\anaconda3\envs\py39\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer u
sing an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

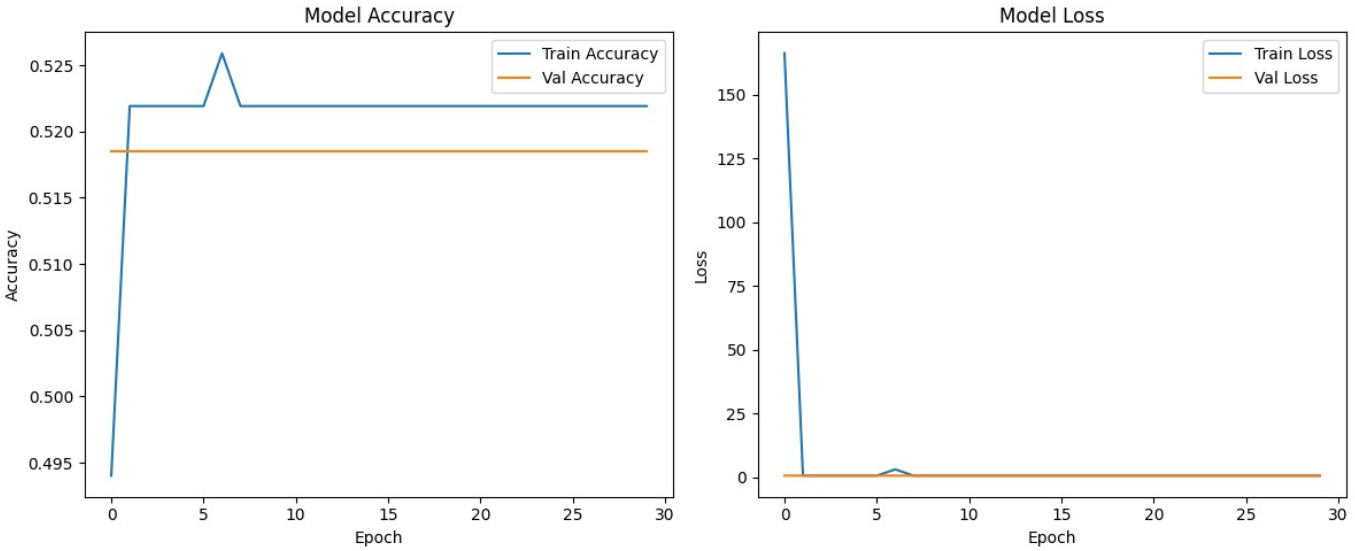
In [25]:
```
model2_binary.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ]
)


start_time = time.time()

# Train the model
history2= model2_binary.fit(
    X_train, y_train,
    epochs=30,
    batch_size=64,
    validation_data=(X_test, y_test)
)
end_time = time.time()
time2 = end_time - start_time
print(f"Training time: {time2:.2f} seconds")
```

```
Epoch 1/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 690ms/step - accuracy: 0.5106 - auc: 0.5341 - loss: 146.8463 - precision: 0.5370 - r
ecall: 0.4138 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall:
1.0000
Epoch 2/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 332ms/step - accuracy: 0.5385 - auc: 0.5000 - loss: 0.6931 - precision: 0.5385 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 3/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 393ms/step - accuracy: 0.5192 - auc: 0.4983 - loss: 0.6931 - precision: 0.5192 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 4/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 329ms/step - accuracy: 0.5431 - auc: 0.5000 - loss: 0.6930 - precision: 0.5431 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 5/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.5020 - auc: 0.5000 - loss: 0.6931 - precision: 0.5020 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 6/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 440ms/step - accuracy: 0.5473 - auc: 0.5000 - loss: 0.6929 - precision: 0.5473 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 7/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 449ms/step - accuracy: 0.5187 - auc: 0.5113 - loss: 3.2830 - precision: 0.5186 - rec
all: 0.8366 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 8/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 451ms/step - accuracy: 0.5385 - auc: 0.5000 - loss: 0.6929 - precision: 0.5385 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5089 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 9/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 380ms/step - accuracy: 0.5254 - auc: 0.5000 - loss: 0.6929 - precision: 0.5254 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 10/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 322ms/step - accuracy: 0.5103 - auc: 0.5000 - loss: 0.6931 - precision: 0.5103 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 11/30
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 284ms/step - accuracy: 0.5489 - auc: 0.5000 - loss: 0.6927 - precision: 0.5489 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 12/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 226ms/step - accuracy: 0.4890 - auc: 0.5000 - loss: 0.6933 - precision: 0.4890 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6930 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 13/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 345ms/step - accuracy: 0.5317 - auc: 0.5000 - loss: 0.6928 - precision: 0.5317 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 14/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 328ms/step - accuracy: 0.5239 - auc: 0.5000 - loss: 0.6929 - precision: 0.5239 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 15/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 339ms/step - accuracy: 0.5161 - auc: 0.5000 - loss: 0.6930 - precision: 0.5161 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 16/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - accuracy: 0.5082 - auc: 0.5000 - loss: 0.6931 - precision: 0.5082 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 17/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 445ms/step - accuracy: 0.5624 - auc: 0.5000 - loss: 0.6923 - precision: 0.5624 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 18/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 364ms/step - accuracy: 0.5197 - auc: 0.5000 - loss: 0.6929 - precision: 0.5197 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 19/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 432ms/step - accuracy: 0.5374 - auc: 0.5000 - loss: 0.6926 - precision: 0.5374 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 20/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.5218 - auc: 0.5000 - loss: 0.6928 - precision: 0.5218 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 21/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 315ms/step - accuracy: 0.5322 - auc: 0.5000 - loss: 0.6927 - precision: 0.5322 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 22/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 329ms/step - accuracy: 0.5108 - auc: 0.5000 - loss: 0.6930 - precision: 0.5108 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 23/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 314ms/step - accuracy: 0.5166 - auc: 0.5000 - loss: 0.6929 - precision: 0.5166 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 24/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 315ms/step - accuracy: 0.4978 - auc: 0.5000 - loss: 0.6932 - precision: 0.4978 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6929 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 25/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 234ms/step - accuracy: 0.5369 - auc: 0.5000 - loss: 0.6925 - precision: 0.5369 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 26/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 245ms/step - accuracy: 0.5593 - auc: 0.5000 - loss: 0.6921 - precision: 0.5593 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 27/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 320ms/step - accuracy: 0.5218 - auc: 0.5000 - loss: 0.6928 - precision: 0.5218 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 28/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 474ms/step - accuracy: 0.4994 - auc: 0.5000 - loss: 0.6932 - precision: 0.4994 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 29/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 440ms/step - accuracy: 0.5556 - auc: 0.5000 - loss: 0.6921 - precision: 0.5556 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Epoch 30/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 453ms/step - accuracy: 0.5135 - auc: 0.5000 - loss: 0.6929 - precision: 0.5135 - rec
all: 1.0000 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 0.6928 - val_precision: 0.5185 - val_recall: 1.
0000
Training time: 50.31 seconds
```

In [26]:
```
plt.figure(figsize=(12, 5))
```

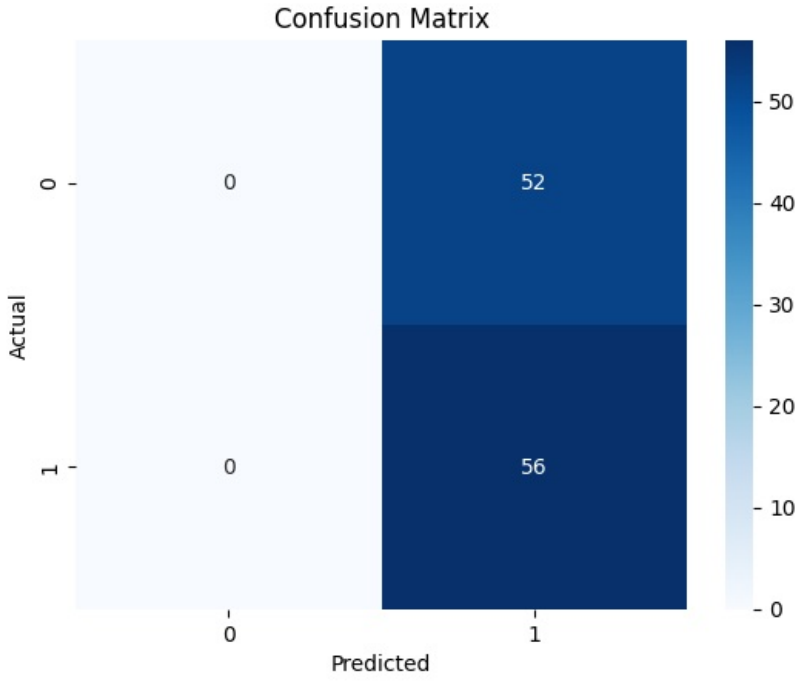
```
plt.subplot(1, 2, 1)
plt.plot(history2.history['accuracy'], label='Train Accuracy')
plt.plot(history2.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history2.history['loss'], label='Train Loss')
plt.plot(history2.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



- The model's training and validation accuracy are both around 52%, meaning it's not learning for all epochs.

In [27]:
```
loss1, accuracy1, precision1, recall1, auc1 = model2_binary.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy1:.4f}")
print(f"Test Precision: {precision1:.4f}")
print(f"Test Recall: {recall1:.4f}")
print(f"Test AUC: {auc1:.4f}")

y_pred_probs = model2_binary.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype("int32").flatten()
y_true = y_test.flatten()

cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.5022 - auc: 0.5000 - loss: 0.6932 - precision: 0.5022 - reca
ll: 1.0000
Test Accuracy: 0.5185
Test Precision: 0.5185
Test Recall: 1.0000
Test AUC: 0.5000
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
```



- All the samples classified as Red-Winged Blackbird suggesting this model is not performing well.

CNN- Model3

In [28]:
```
# Defining model
model3_binary = Sequential()
model3_binary.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model3_binary.add(MaxPooling2D(pool_size=(2, 2)))

model3_binary.add(Conv2D(64, (3, 3), activation='relu'))
model3_binary.add(MaxPooling2D(pool_size=(2, 2)))

model3_binary.add(Flatten())
model3_binary.add(Dense(128, activation='relu'))
model3_binary.add(Dropout(0.3))
model3_binary.add(Dense(1, activation='sigmoid'))
```

```
C:\Users\NIDHI KAPADIYA\anaconda3\envs\py39\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [29]:
```
model3_binary.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ]
)

start_time = time.time()

history3= model3_binary.fit(
    X_train, y_train,
    epochs=30,
    batch_size=64,
```

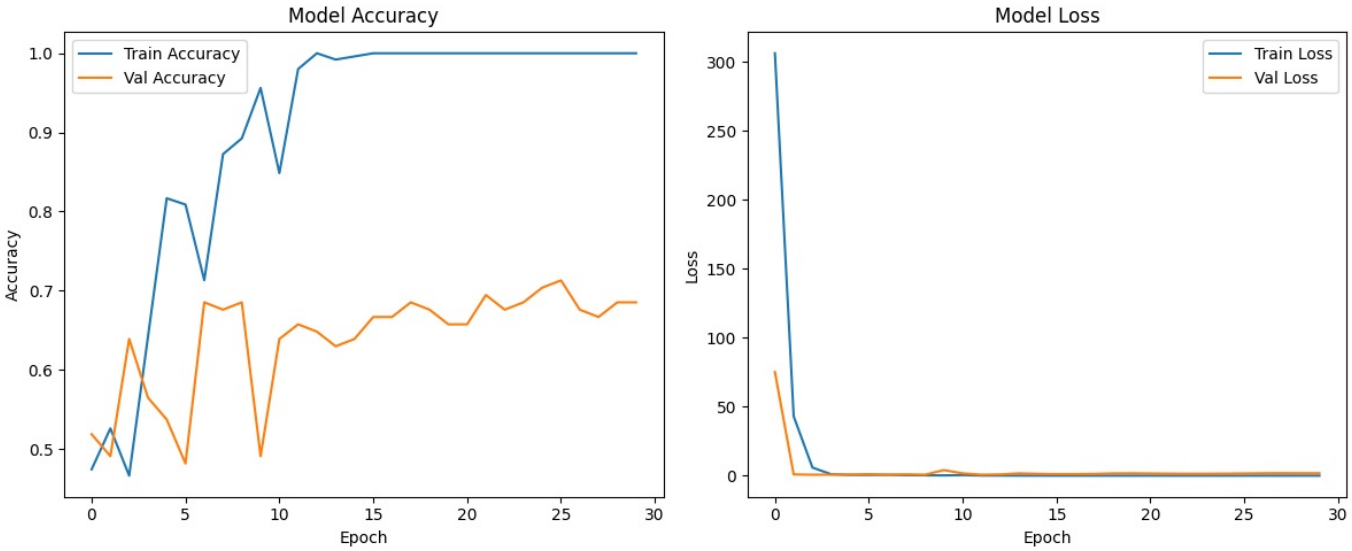
```
    validation_data=(X_test, y_test)
)
end_time = time.time()
time3 = end_time - start_time
print(f"Training time: {time3:.2f} seconds")
```

```
Epoch 1/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 19s 4s/step - accuracy: 0.4917 - auc: 0.5135 - loss: 296.4748 - precision: 0.5266 - rec
all: 0.4075 - val_accuracy: 0.5185 - val_auc: 0.5000 - val_loss: 75.1613 - val_precision: 0.5185 - val_recall: 1
.0000
Epoch 2/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.5249 - auc: 0.5265 - loss: 56.5760 - precision: 0.5194 - reca
ll: 0.9095 - val_accuracy: 0.4907 - val_auc: 0.5019 - val_loss: 0.8928 - val_precision: 0.6000 - val_recall: 0.0
536
Epoch 3/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.4385 - auc: 0.4686 - loss: 8.0707 - precision: 0.4917 - recal
l: 0.3801 - val_accuracy: 0.6389 - val_auc: 0.7160 - val_loss: 0.6668 - val_precision: 0.6104 - val_recall: 0.83
93
Epoch 4/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.6946 - auc: 0.6854 - loss: 0.8393 - precision: 0.6713 - recal
l: 0.8223 - val_accuracy: 0.5648 - val_auc: 0.6690 - val_loss: 0.7297 - val_precision: 0.7368 - val_recall: 0.25
00
Epoch 5/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.8277 - auc: 0.9082 - loss: 0.5156 - precision: 0.8769 - recall
: 0.7821 - val_accuracy: 0.5370 - val_auc: 0.6784 - val_loss: 0.7821 - val_precision: 0.8750 - val_recall: 0.125
0
Epoch 6/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.7480 - auc: 0.8430 - loss: 0.5499 - precision: 0.8437 - recall
: 0.6606 - val_accuracy: 0.4815 - val_auc: 0.7296 - val_loss: 1.1133 - val_precision: 0.0000e+00 - val_recall: 0
.0000e+00
Epoch 7/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.6530 - auc: 0.7938 - loss: 0.6571 - precision: 0.7748 - recall
: 0.5999 - val_accuracy: 0.6852 - val_auc: 0.6575 - val_loss: 0.6804 - val_precision: 0.6719 - val_recall: 0.767
9
Epoch 8/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9094 - auc: 0.9647 - loss: 0.3482 - precision: 0.8885 - recall
: 0.9406 - val_accuracy: 0.6759 - val_auc: 0.6581 - val_loss: 0.9746 - val_precision: 0.6296 - val_recall: 0.910
7
Epoch 9/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8736 - auc: 0.9812 - loss: 0.2767 - precision: 0.8101 - recall
: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.6793 - val_loss: 0.7082 - val_precision: 0.6667 - val_recall: 0.785
7
Epoch 10/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9601 - auc: 0.9922 - loss: 0.1798 - precision: 0.9320 - recall
: 0.9969 - val_accuracy: 0.4907 - val_auc: 0.6044 - val_loss: 4.0195 - val_precision: 1.0000 - val_recall: 0.017
9
Epoch 11/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7811 - auc: 0.9194 - loss: 0.6985 - precision: 0.9061 - recall
: 0.6608 - val_accuracy: 0.6389 - val_auc: 0.7385 - val_loss: 1.6781 - val_precision: 1.0000 - val_recall: 0.303
6
Epoch 12/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9634 - auc: 0.9967 - loss: 0.1097 - precision: 1.0000 - recall
: 0.9254 - val_accuracy: 0.6574 - val_auc: 0.7109 - val_loss: 0.6994 - val_precision: 0.6727 - val_recall: 0.660
7
Epoch 13/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0834 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6481 - val_auc: 0.6884 - val_loss: 0.9071 - val_precision: 0.6667 - val_recall: 0.642
9
Epoch 14/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.9968 - auc: 1.0000 - loss: 0.0434 - precision: 0.9940 - recal
l: 1.0000 - val_accuracy: 0.6296 - val_auc: 0.7148 - val_loss: 1.6698 - val_precision: 0.7857 - val_recall: 0.39
29
Epoch 15/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9927 - auc: 0.9994 - loss: 0.0311 - precision: 1.0000 - recall
: 0.9850 - val_accuracy: 0.6389 - val_auc: 0.7373 - val_loss: 1.3296 - val_precision: 0.7576 - val_recall: 0.446
4
Epoch 16/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0161 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7335 - val_loss: 1.1323 - val_precision: 0.7174 - val_recall: 0.589
3
Epoch 17/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 3s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0130 - precision: 1.0000 - recal
l: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7399 - val_loss: 1.1568 - val_precision: 0.6923 - val_recall: 0.64
29
Epoch 18/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0111 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.7625 - val_loss: 1.2971 - val_precision: 0.7750 - val_recall: 0.553
6
Epoch 19/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0045 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7490 - val_loss: 1.6417 - val_precision: 0.8387 - val_recall: 0.464
3
Epoch 20/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0040 - precision: 1.0000 - recall
```

```
: 1.0000 - val_accuracy: 0.6574 - val_auc: 0.7643 - val_loss: 1.7229 - val_precision: 0.8065 - val_recall: 0.446
4
Epoch 21/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0022 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6574 - val_auc: 0.7577 - val_loss: 1.5974 - val_precision: 0.7568 - val_recall: 0.500
0
Epoch 22/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0015 - precision: 1.0000 - recall
: 1.0000 - val_accuracy: 0.6944 - val_auc: 0.7512 - val_loss: 1.4660 - val_precision: 0.7447 - val_recall: 0.625
0
Epoch 23/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 7.9237e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7450 - val_loss: 1.3970 - val_precision: 0.6981 - val_recall: 0
.6607
Epoch 24/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 4.5374e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.7471 - val_loss: 1.4206 - val_precision: 0.7037 - val_recall: 0
.6786
Epoch 25/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 2.2345e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.7037 - val_auc: 0.7533 - val_loss: 1.4764 - val_precision: 0.7400 - val_recall: 0
.6607
Epoch 26/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 3.6317e-04 - precision: 1.0000 - r
ecall: 1.0000 - val_accuracy: 0.7130 - val_auc: 0.7596 - val_loss: 1.6019 - val_precision: 0.7660 - val_recall:
0.6429
Epoch 27/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 2.8218e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6759 - val_auc: 0.7643 - val_loss: 1.7354 - val_precision: 0.7442 - val_recall: 0
.5714
Epoch 28/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 1.9998e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7555 - val_loss: 1.8115 - val_precision: 0.7381 - val_recall: 0
.5536
Epoch 29/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 7.7496e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.7538 - val_loss: 1.7517 - val_precision: 0.7391 - val_recall: 0
.6071
Epoch 30/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 1.6225e-04 - precision: 1.0000 - re
call: 1.0000 - val_accuracy: 0.6852 - val_auc: 0.7577 - val_loss: 1.7582 - val_precision: 0.7292 - val_recall: 0
.6250
Training time: 270.09 seconds
```

In [39]:

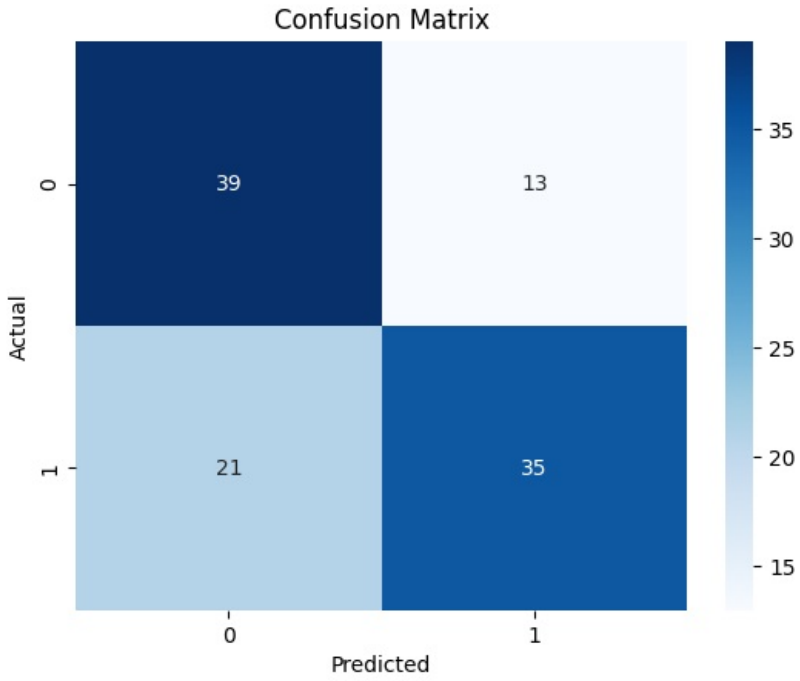
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history3.history['accuracy'], label='Train Accuracy')
plt.plot(history3.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history3.history['loss'], label='Train Loss')
plt.plot(history3.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



- The model reaches nearly perfect training accuracy, but validation accuracy stays around 0.65-0.70 and becomes plateau after 25.
- This means the model is overfitting — it learns the training data too well but doesn't generalize to new data.

In [31]:
```
loss1, accuracy1, precision1, recall1, auc1 = model3_binary.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy1:.4f}")
print(f"Test Precision: {precision1:.4f}")
print(f"Test Recall: {recall1:.4f}")
print(f"Test AUC: {auc1:.4f}")

y_pred_probs = model3_binary.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype("int32").flatten()
y_true = y_test.flatten()

cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 183ms/step - accuracy: 0.7189 - auc: 0.7684 - loss: 1.8712 - precision: 0.7607 - rec
all: 0.6468
Test Accuracy: 0.6852
Test Precision: 0.7292
Test Recall: 0.6250
Test AUC: 0.7577
WARNING:tensorflow:5 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_s
tep_on_data_distributed at 0x0000017C7E6321F0> triggered tf.function retracing. Tracing is expensive and the exc
essive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors wi
th different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function out
side of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. F
or (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 12 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_
step_on_data_distributed at 0x0000017C7E6321F0> triggered tf.function retracing. Tracing is expensive and the ex
cessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function ou
tside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorf
low.org/api_docs/python/tf/function for  more details.
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 203ms/step
```



- The heatmap shows that the model correctly predicted most American Robin (39).
- However, some American Robin samples were misclassified as Red-winged Blackbird (13).

Tunning Model- CNN Model1

- Out of three cnn model, model1 and model3 performed well.
- So, I am tunning that models atarting with model1 which was good in predicting Red-Winged Blakbird.
- Trying with three different batch sizes and 2 different dropout rates.
- Keeping epoch 15, as we seen from first CNN model1 that it is not improving after 15 epochs.

In [33]:
```
import time
from tensorflow.keras import backend as K
import gc

# Trying 3 different batch sizes and 2 different dropout rates
batch_sizes = [16, 32, 64]
dropout_rates = [0.2, 0.4]

results1 = []
history_records1 = {}

for batch_size in batch_sizes:
    for dropout_rate in dropout_rates:
        model_name = f"bs{batch_size}_do{int(dropout_rate*10)}"
        print(f"\nTraining with batch_size={batch_size}, dropout_rate={dropout_rate}")

        # Clear session and memory
        K.clear_session()
        gc.collect()

        # Build the model
        model = Sequential()
```

```
        model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(32, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(64, activation='relu'))
        model.add(Dropout(dropout_rate))
        model.add(Dense(1, activation='sigmoid'))

        model.compile(
            optimizer='adam',
            loss='binary_crossentropy',
            metrics=[
                'accuracy',
                tf.keras.metrics.Precision(name='precision'),
                tf.keras.metrics.Recall(name='recall'),
                tf.keras.metrics.AUC(name='auc')
            ]
        )


        start_time = time.time()

        history5 = model.fit(
            X_train, y_train,
            epochs=15,
            batch_size=batch_size,
            validation_data=(X_test, y_test),
            verbose=0
        )

        elapsed_time = time.time() - start_time
        print(f"Training time: {elapsed_time:.2f} seconds")


        history_records1[model_name] = history5.history
        scores = model.evaluate(X_test, y_test, verbose=0)
        results1.append({
            'model': model_name,
            'batch_size': batch_size,
            'dropout_rate': dropout_rate,
            'val_accuracy': scores[1],
            'val_precision': scores[2],
            'val_recall': scores[3],
            'val_auc': scores[4],
            'training_time_sec': elapsed_time
        })
```

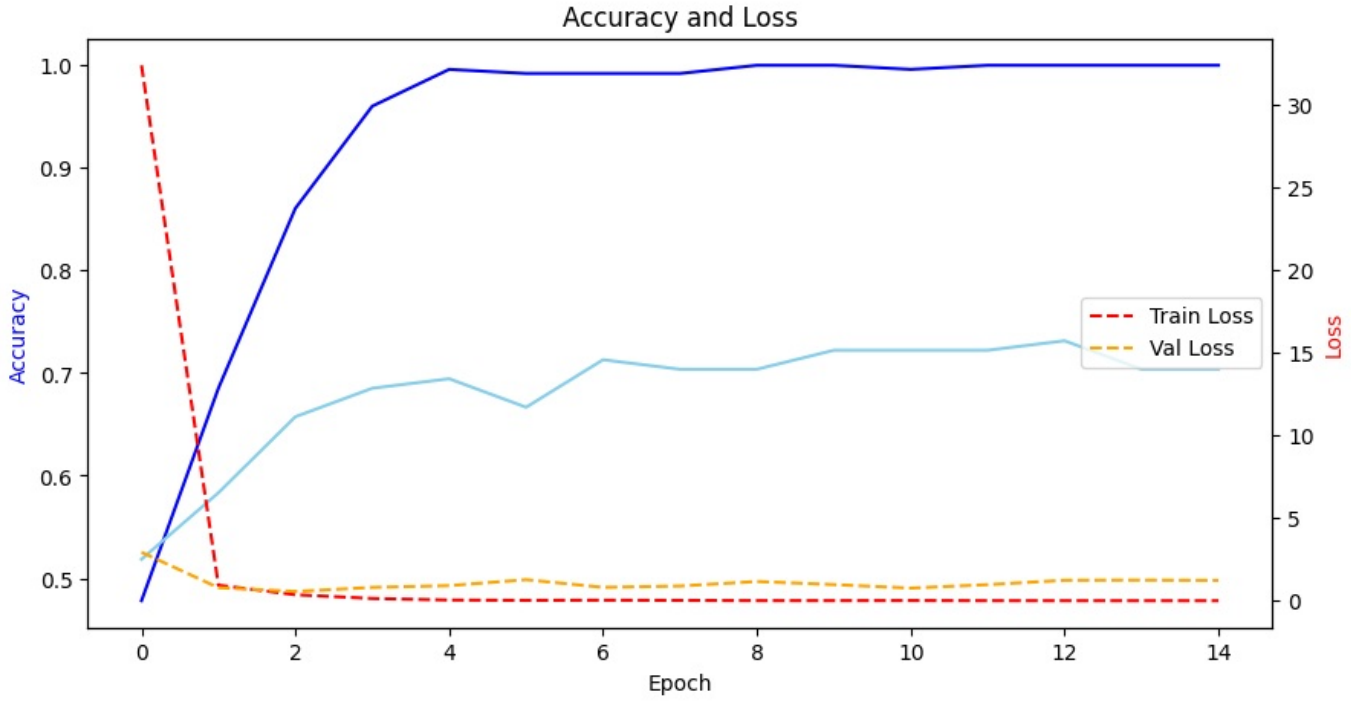
```
Training with batch_size=16, dropout_rate=0.2
Training time: 53.83 seconds

Training with batch_size=16, dropout_rate=0.4
Training time: 54.83 seconds

Training with batch_size=32, dropout_rate=0.2
Training time: 43.24 seconds

Training with batch_size=32, dropout_rate=0.4
Training time: 42.31 seconds

Training with batch_size=64, dropout_rate=0.2
Training time: 40.77 seconds

Training with batch_size=64, dropout_rate=0.4
Training time: 38.44 seconds
```

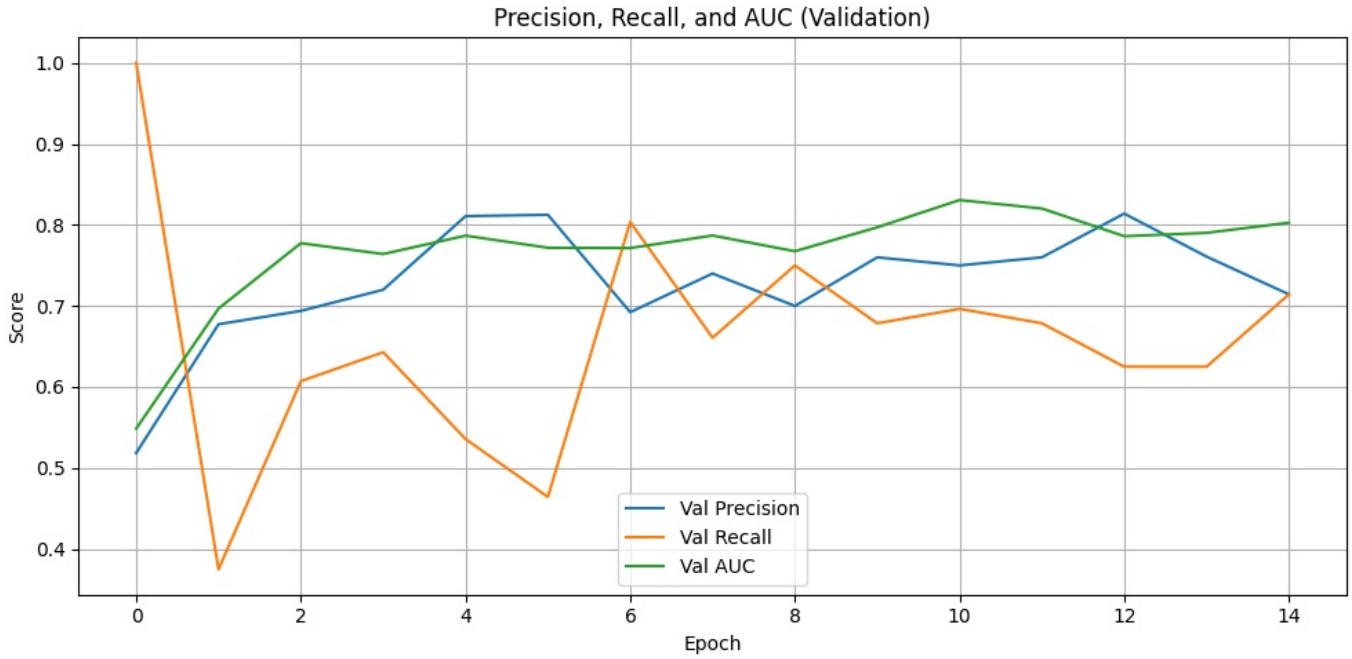
- All variaiation of CNN model1 trained under 1 minute, which is reasonable time.

In [35]:
```
# Print results sorted by validation accuracy
results_sorted = sorted(results1, key=lambda x: x['val_accuracy'], reverse=True)
for r in results_sorted:
    print(r)
```

{'model': 'bs16_do4', 'batch_size': 16, 'dropout_rate': 0.4, 'val_accuracy': 0.7037037014961243, 'val_precision': 0.7142857313156128, 'val_recall': 0.7142857313156128, 'val_auc': 0.8027129173278809, 'training_time_sec': 54.82879424095154}
{'model': 'bs32_do4', 'batch_size': 32, 'dropout_rate': 0.4, 'val_accuracy': 0.6944444179534912, 'val_precision': 0.7804877758026123, 'val_recall': 0.5714285969734192, 'val_auc': 0.7723214626312256, 'training_time_sec': 42.31474852561951}
{'model': 'bs16_do2', 'batch_size': 16, 'dropout_rate': 0.2, 'val_accuracy': 0.6388888955116272, 'val_precision': 0.6491228342056274, 'val_recall': 0.6607142686843872, 'val_auc': 0.7699175477027893, 'training_time_sec': 53.829193115234375}
{'model': 'bs32_do2', 'batch_size': 32, 'dropout_rate': 0.2, 'val_accuracy': 0.6296296119689941, 'val_precision': 0.6818181872367859, 'val_recall': 0.5357142686843872, 'val_auc': 0.7032967209815979, 'training_time_sec': 43.23538064956665}
{'model': 'bs64_do4', 'batch_size': 64, 'dropout_rate': 0.4, 'val_accuracy': 0.5092592835426331, 'val_precision': 0.514018714427948, 'val_recall': 0.9821428656578064, 'val_auc': 0.4910714328289032, 'training_time_sec': 38.43986797332764}
{'model': 'bs64_do2', 'batch_size': 64, 'dropout_rate': 0.2, 'val_accuracy': 0.48148149251937866, 'val_precision': 0.0, 'val_recall': 0.0, 'val_auc': 0.509615421295166, 'training_time_sec': 40.77443170547485}

In [46]:
```
best_history=history_records1['bs16_do4']
plt.figure(figsize=(10, 5))
plt.title('Accuracy and Loss')
plt.xlabel('Epoch')

plt.plot(best_history['accuracy'], label='Train Accuracy', color='blue')
plt.plot(best_history['val_accuracy'], label='Val Accuracy', color='skyblue')

plt.ylabel('Accuracy', color='blue')
plt.twinx()
plt.plot(best_history['loss'], label='Train Loss', color='red', linestyle='--')
plt.plot(best_history['val_loss'], label='Val Loss', color='orange', linestyle='--')
plt.ylabel('Loss', color='red')
plt.legend(loc='center right')
plt.show()


plt.figure(figsize=(10, 5))
plt.plot(best_history['val_precision'], label='Val Precision')
plt.plot(best_history['val_recall'], label='Val Recall')
plt.plot(best_history['val_auc'], label='Val AUC')
plt.title('Precision, Recall, and AUC (Validation)')
plt.xlabel('Epoch')
plt.ylabel('Score')
plt.legend(loc='lower center')
plt.grid(True)
plt.tight_layout()
plt.show()
```





- Validation accuracy is around 0.7, and AUC at 0.80, meaning the model predicts both classes well.

In [37]:
```
import time
# Grid of batch sizes and dropout rates
batch_sizes = [16, 32, 64]
dropout_rates = [0.2, 0.4]

results2 = []
history_records2 = {}

for batch_size in batch_sizes:
    for dropout_rate in dropout_rates:
        model_name = f"bs{batch_size}_do{int(dropout_rate*10)}"
        print(f"Training with batch_size={batch_size}, dropout_rate={dropout_rate}")

        # Clear session and memory
        K.clear_session()
        gc.collect()

        # Build the model
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(dropout_rate))
        model.add(Dense(1, activation='sigmoid'))

        model.compile(
            optimizer='adam',
            loss='binary_crossentropy',
            metrics=[
                'accuracy',
                tf.keras.metrics.Precision(name='precision'),
                tf.keras.metrics.Recall(name='recall'),
                tf.keras.metrics.AUC(name='auc')
            ]
        )

        # Measure training time
        start_time = time.time()

        history = model.fit(
            X_train, y_train,
```

```
            epochs=15,
            batch_size=batch_size,
            validation_data=(X_test, y_test),
            verbose=0
        )

        elapsed_time = time.time() - start_time
        print(f"Training time: {elapsed_time:.2f} seconds")

        # Save results
        history_records2[model_name] = history.history
        scores = model.evaluate(X_test, y_test, verbose=0)
        results2.append({
            'config': model_name,
            'batch_size': batch_size,
            'dropout_rate': dropout_rate,
            'val_accuracy': scores[1],
            'val_precision': scores[2],
            'val_recall': scores[3],
            'val_auc': scores[4],
            'training_time_sec': elapsed_time
        })
```

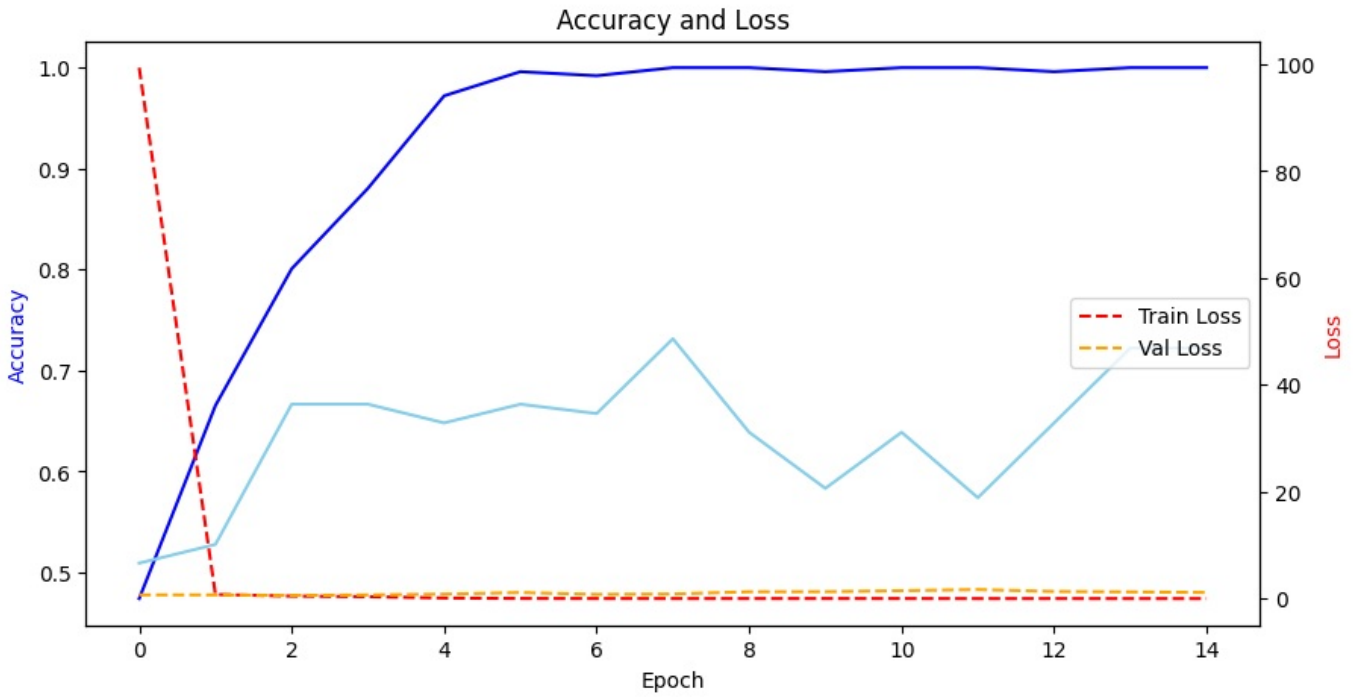
```
Training with batch_size=16, dropout_rate=0.2
Training time: 155.51 seconds
Training with batch_size=16, dropout_rate=0.4
Training time: 158.51 seconds
Training with batch_size=32, dropout_rate=0.2
Training time: 141.91 seconds
Training with batch_size=32, dropout_rate=0.4
Training time: 137.51 seconds
Training with batch_size=64, dropout_rate=0.2
Training time: 125.60 seconds
Training with batch_size=64, dropout_rate=0.4
Training time: 128.76 seconds
```

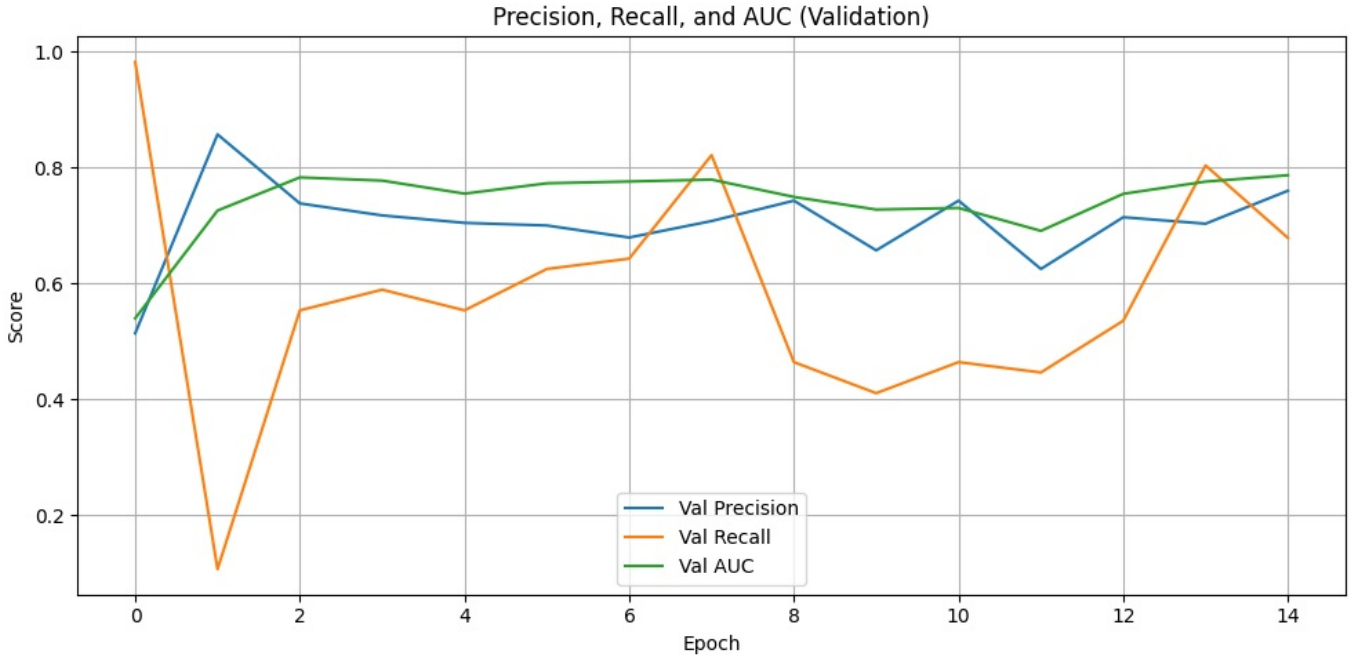
In [38]:
```
# Print results sorted by validation accuracy
results_sorted2 = sorted(results2, key=lambda x: x['val_accuracy'], reverse=True)
for r in results_sorted2:
    print(r)
```

```
{'config': 'bs16_do4', 'batch_size': 16, 'dropout_rate': 0.4, 'val_accuracy': 0.7222222089767456, 'val_precision
': 0.7599999904632568, 'val_recall': 0.6785714030265808, 'val_auc': 0.7867444753646851, 'training_time_sec': 158
.51228523254395}
{'config': 'bs64_do2', 'batch_size': 64, 'dropout_rate': 0.2, 'val_accuracy': 0.7129629850387573, 'val_precision
': 0.7358490824699402, 'val_recall': 0.6964285969734192, 'val_auc': 0.776785671710968, 'training_time_sec': 125.
60413551330566}
{'config': 'bs32_do4', 'batch_size': 32, 'dropout_rate': 0.4, 'val_accuracy': 0.6944444179534912, 'val_precision
': 0.7446808218955994, 'val_recall': 0.625, 'val_auc': 0.7251030206680298, 'training_time_sec': 137.505529880523
68}
{'config': 'bs64_do4', 'batch_size': 64, 'dropout_rate': 0.4, 'val_accuracy': 0.6574074029922485, 'val_precision
': 0.7021276354789734, 'val_recall': 0.5892857313156128, 'val_auc': 0.738839328289032, 'training_time_sec': 128.
75587511062622}
{'config': 'bs16_do2', 'batch_size': 16, 'dropout_rate': 0.2, 'val_accuracy': 0.6111111044883728, 'val_precision
': 0.6944444179534912, 'val_recall': 0.4464285671710968, 'val_auc': 0.7141139507293701, 'training_time_sec': 155
.51334524154663}
{'config': 'bs32_do2', 'batch_size': 32, 'dropout_rate': 0.2, 'val_accuracy': 0.6018518805503845, 'val_precision
': 0.6857143044471741, 'val_recall': 0.4285714328289032, 'val_auc': 0.7367788553237915, 'training_time_sec': 141
.91226601600647}
```

In [49]:
```
best_history=history_records2['bs16_do4']
plt.figure(figsize=(10, 5))
plt.title('Accuracy and Loss')
plt.xlabel('Epoch')

plt.plot(best_history['accuracy'], label='Train Accuracy', color='blue')
plt.plot(best_history['val_accuracy'], label='Val Accuracy', color='skyblue')

plt.ylabel('Accuracy', color='blue')
plt.twinx()
plt.plot(best_history['loss'], label='Train Loss', color='red', linestyle='--')
plt.plot(best_history['val_loss'], label='Val Loss', color='orange', linestyle='--')
plt.ylabel('Loss', color='red')
plt.legend(loc='center right')
plt.show()


plt.figure(figsize=(10, 5))
plt.plot(best_history['val_precision'], label='Val Precision')
plt.plot(best_history['val_recall'], label='Val Recall')
plt.plot(best_history['val_auc'], label='Val AUC')
plt.title('Precision, Recall, and AUC (Validation)')
plt.xlabel('Epoch')
plt.ylabel('Score')
```

```
plt.legend(loc='lower center')
plt.grid(True)
plt.tight_layout()
plt.show()
```





- Accuracy is around 72% and AUC about 0.78 meaning this model also predicts both class well.

- Out of all model, CNN model2 achieved 72% acuuracy with batch size 16 and dropout rate 0.4.

In [1]:
```
from tensorflow.keras import backend as K
import gc
K.clear_session()
gc.collect()
```

```
WARNING:tensorflow:From C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\backend\common\global_st
ate.py:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.
```

Out[1]: 0

Multiclass classification

Step1: Data Preparation

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.utils import class_weight
from tensorflow.keras.metrics import Precision, Recall, AUC
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

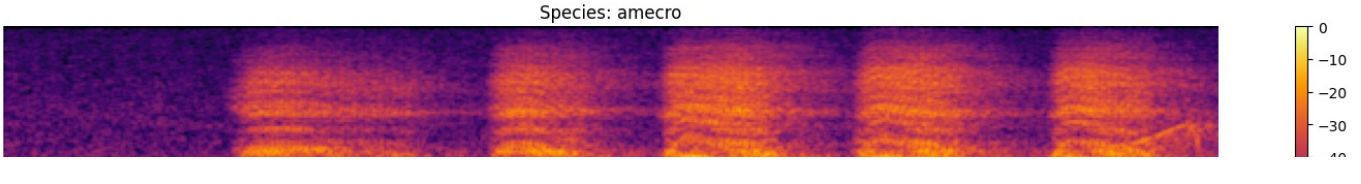
In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

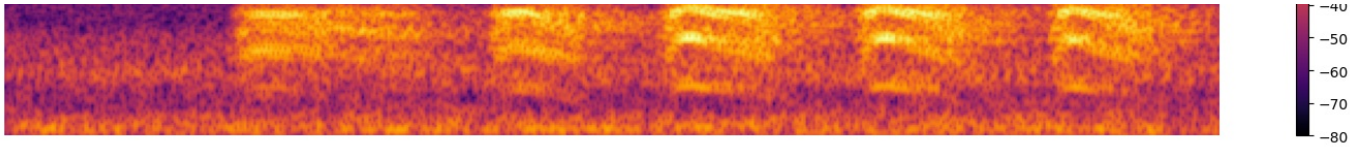
In [3]:
```
# Load the HDF5 file
bird_sound = h5py.File("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\\Homework3\\bi
```

In [4]:
```
species= list(bird_sound.keys())

# Load all spectrograms and labels
X = []
y = []

for label in species:
    data =bird_sound[label][:]
    data = np.transpose(data, (2, 0, 1))
    X.append(data)
    y += [label] * data.shape[0]


X = np.concatenate(X, axis=0)
print(X.shape)
```

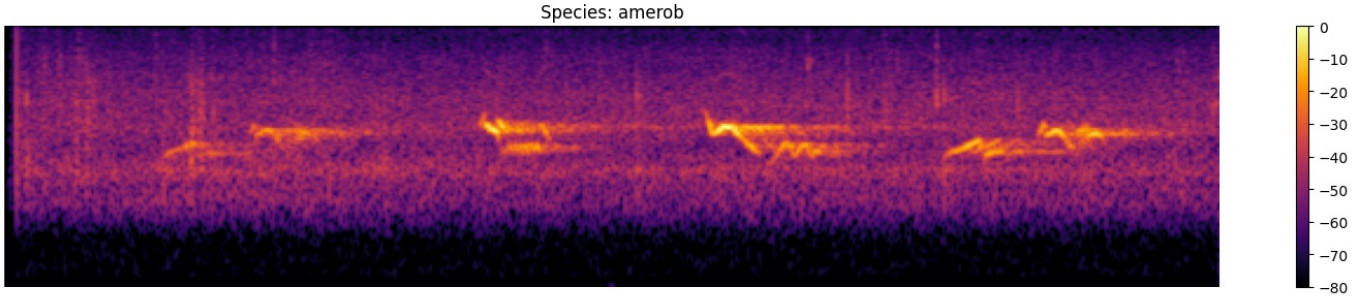
```
(1981, 128, 517)
```

In [5]:
```
# Plot one spectrogram sample from each species
unique_species = list(np.unique(y))
sample_specs = {}

for label in unique_species:
    label_index = y.index(label) if isinstance(y, list) else np.where(np.array(y) == label)[0][0]
    sample = X[label_index, :, :]

    sample_specs[label] = sample

plt.figure(figsize=(15, 3 * len(sample_specs)))

for i, (label, spec) in enumerate(sample_specs.items()):
    plt.subplot(len(sample_specs), 1, i + 1)
    plt.imshow(spec, aspect='auto', origin='lower', cmap='inferno')
    plt.title(f"Species: {label}")
    plt.colorbar()
    plt.axis('off')

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

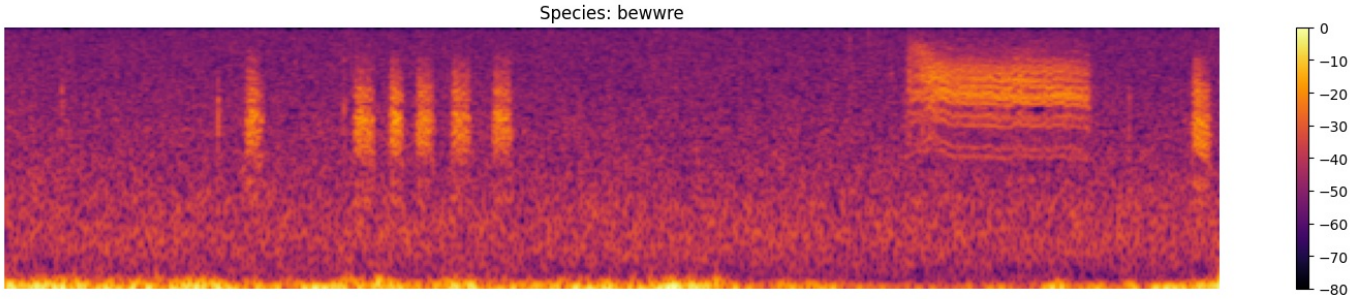




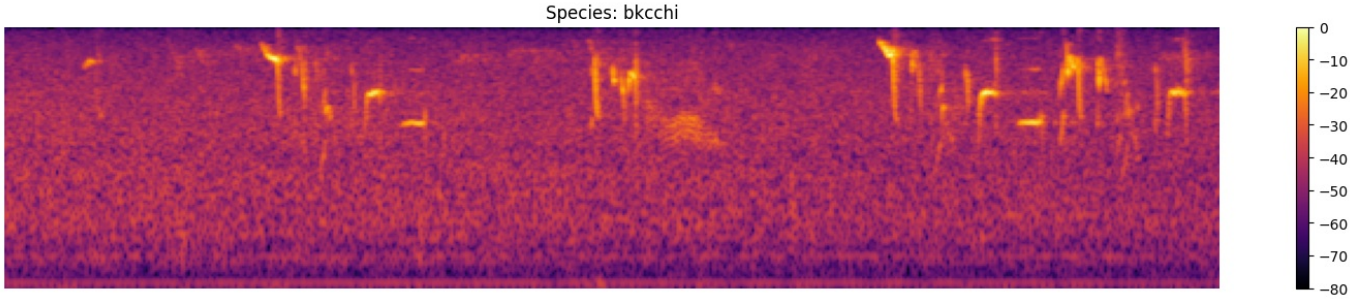
Species: amerob



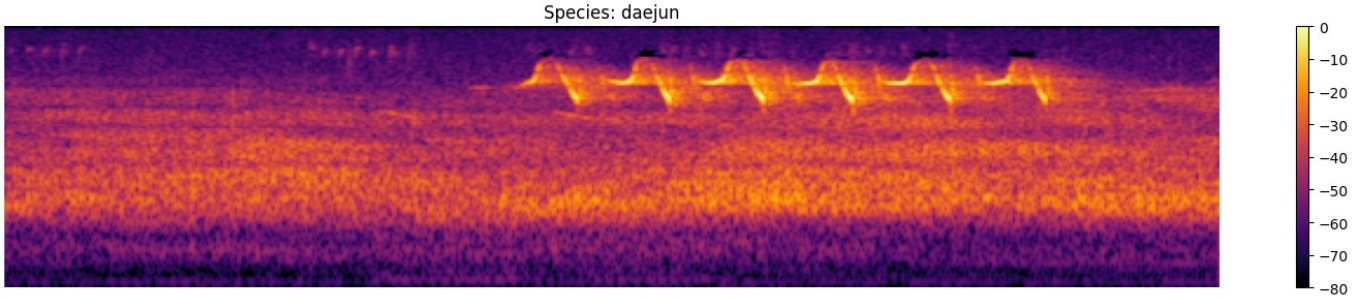
Species: bewwre



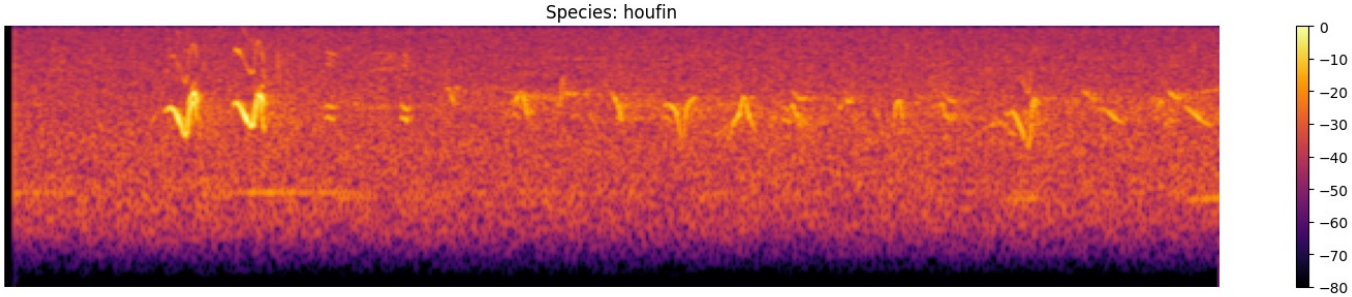
Species: bkcchi



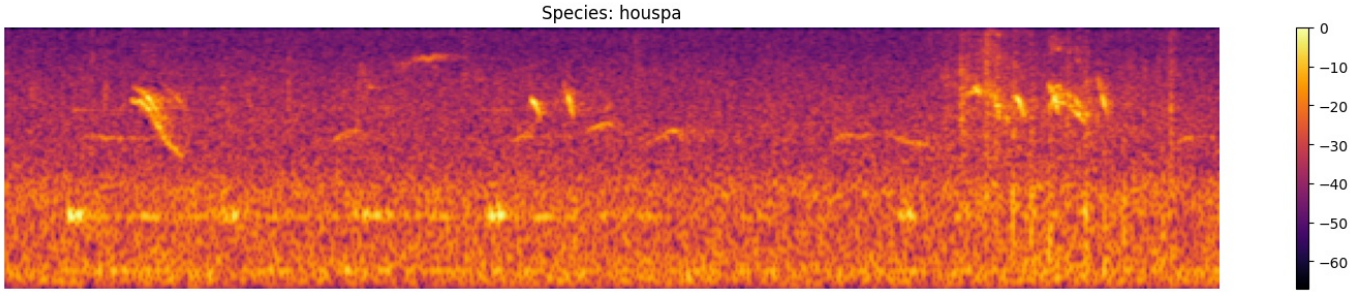
Species: daejun



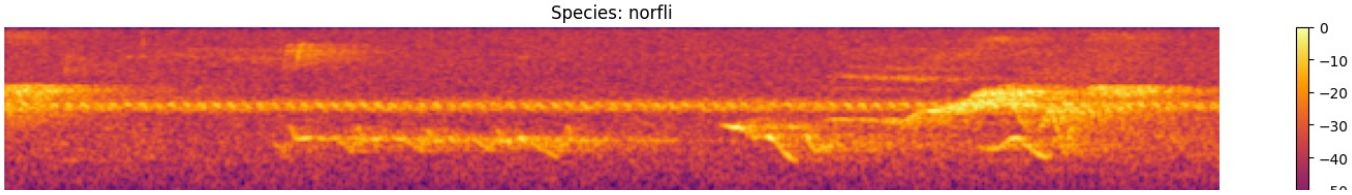
Species: houfin

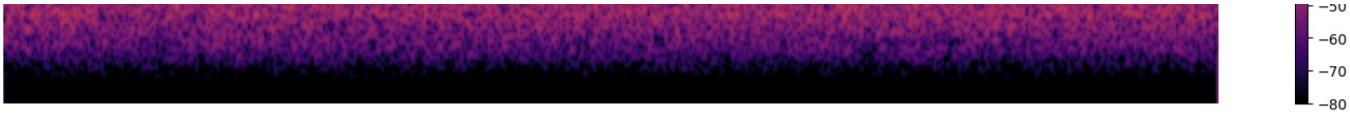


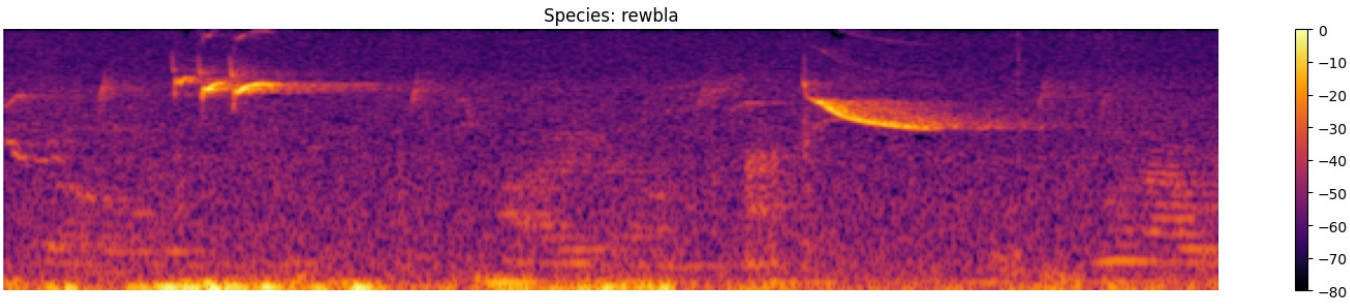
Species: houspa

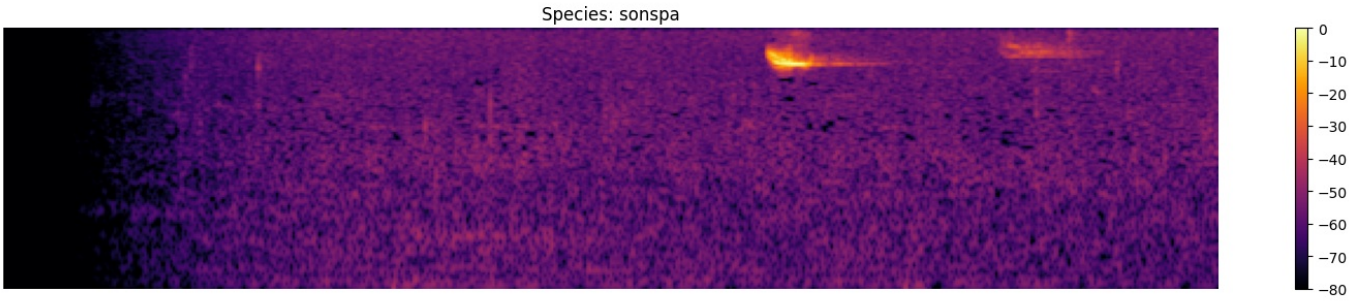


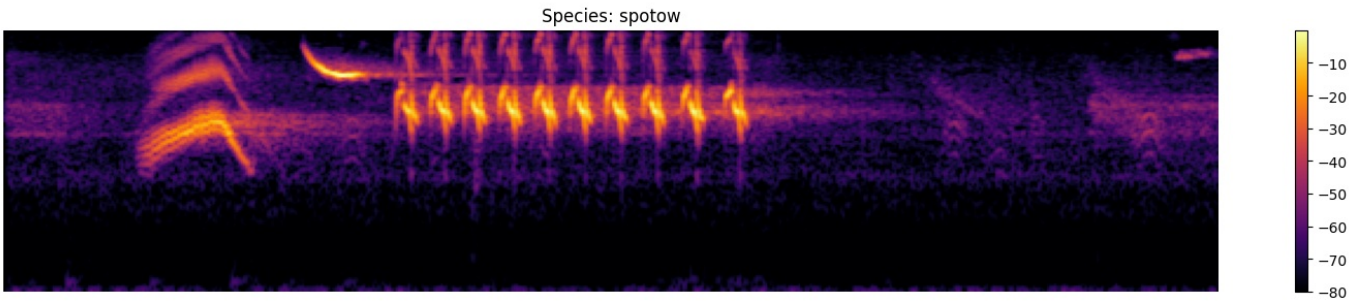
Species: norfli

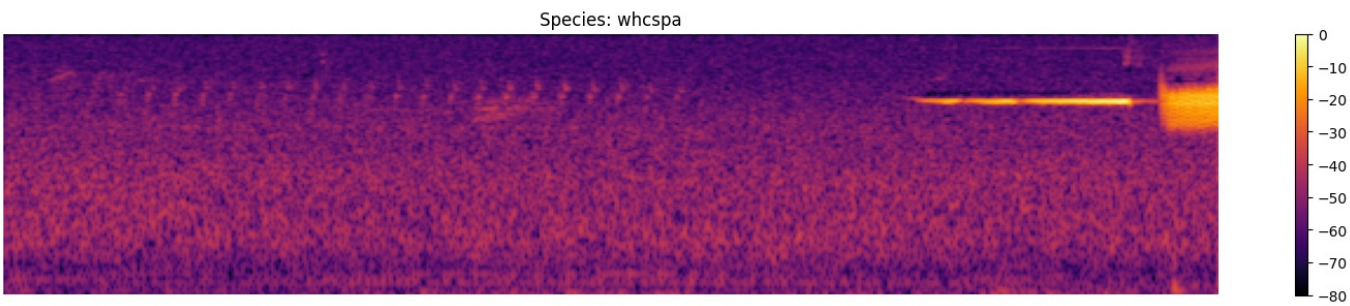












Step2: Train-Test Split

In [6]:
```
X = X.reshape((1981, 128, 517, 1))
y = np.array(y)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_categorical, test_size=0.2, random_state=156, stratify=y_encoded
)
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
print(f"Number of classes: {len(label_encoder.classes_)}")
```

```
X_train shape: (1584, 128, 517, 1), y_train shape: (1584, 12)
X_test shape: (397, 128, 517, 1), y_test shape: (397, 12)
Number of classes: 12
```

Step3: Model Training

Model1

In [8]:
```
# Definigng model1
np.random.seed(123)
tf.random.set_seed(123)

model1_multi = Sequential()
```

```
model1_multi.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model1_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1_multi.add(Conv2D(64, (3, 3), activation='relu'))
model1_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1_multi.add(Flatten())
model1_multi.add(Dense(128, activation='relu'))
model1_multi.add(Dropout(0.5))

model1_multi.add(Dense(12, activation='softmax'))


model1_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [9]:

```
import time
start_time = time.time()

history1 = model1_multi.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=256
)

end_time = time.time()
time1 = end_time - start_time
print(f"\nTraining time: {time1:.2f} seconds")
```

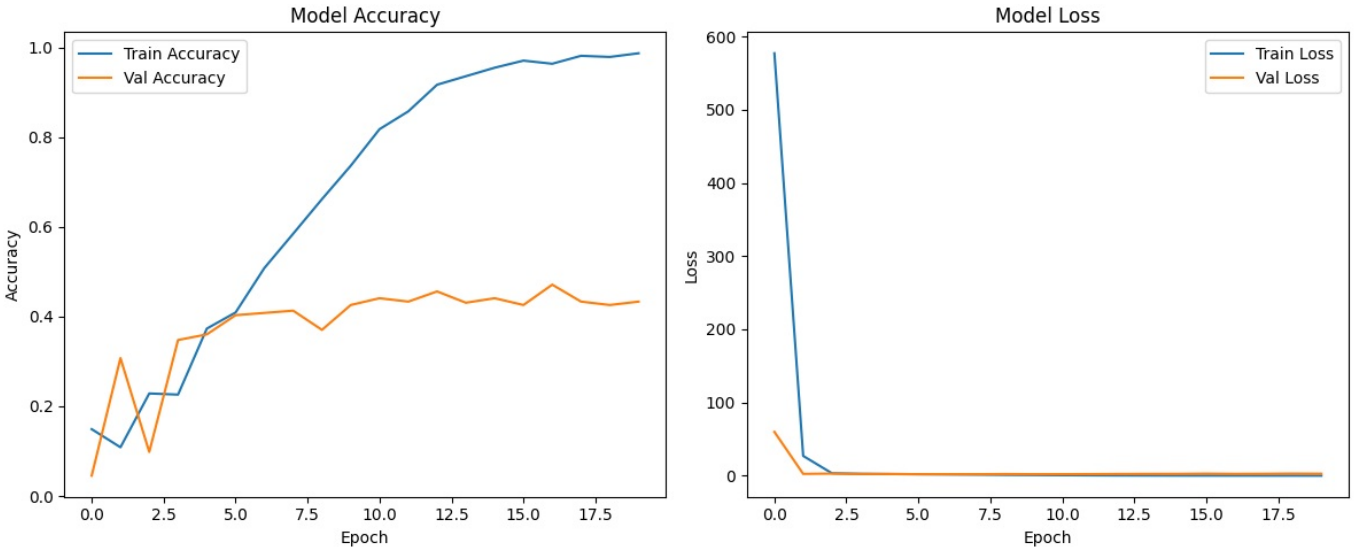
```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 34s 5s/step - accuracy: 0.1411 - auc: 0.5307 - loss: 545.9827 - precision: 0.1426 - rec
all: 0.1411 - val_accuracy: 0.0453 - val_auc: 0.4780 - val_loss: 59.9108 - val_precision: 0.0455 - val_recall: 0
.0453
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.1031 - auc: 0.5173 - loss: 44.2694 - precision: 0.1022 - reca
ll: 0.0866 - val_accuracy: 0.3073 - val_auc: 0.5002 - val_loss: 2.4888 - val_precision: 0.0000e+00 - val_recall:
0.0000e+00
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 34s 5s/step - accuracy: 0.2602 - auc: 0.5037 - loss: 3.2944 - precision: 0.0763 - recal
l: 0.0186 - val_accuracy: 0.0982 - val_auc: 0.4269 - val_loss: 2.9123 - val_precision: 0.0000e+00 - val_recall:
0.0000e+00
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.1834 - auc: 0.5233 - loss: 2.7006 - precision: 0.0000e+00 - r
ecall: 0.0000e+00 - val_accuracy: 0.3476 - val_auc: 0.6964 - val_loss: 2.4225 - val_precision: 0.0000e+00 - val_
recall: 0.0000e+00
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.3829 - auc: 0.7179 - loss: 2.3431 - precision: 0.3555 - recal
l: 0.0182 - val_accuracy: 0.3602 - val_auc: 0.7486 - val_loss: 2.2619 - val_precision: 1.0000 - val_recall: 0.00
25
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.4012 - auc: 0.7967 - loss: 2.0732 - precision: 0.8099 - recal
l: 0.0506 - val_accuracy: 0.4030 - val_auc: 0.7713 - val_loss: 2.0368 - val_precision: 0.5932 - val_recall: 0.08
82
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.4880 - auc: 0.8424 - loss: 1.7971 - precision: 0.6669 - recal
l: 0.1303 - val_accuracy: 0.4081 - val_auc: 0.7851 - val_loss: 2.0887 - val_precision: 0.5025 - val_recall: 0.25
69
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 4s/step - accuracy: 0.5516 - auc: 0.8821 - loss: 1.5779 - precision: 0.7027 - recal
l: 0.2674 - val_accuracy: 0.4131 - val_auc: 0.7977 - val_loss: 1.9996 - val_precision: 0.5183 - val_recall: 0.21
41
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 33s 5s/step - accuracy: 0.6463 - auc: 0.9291 - loss: 1.2091 - precision: 0.7768 - recal
l: 0.4323 - val_accuracy: 0.3703 - val_auc: 0.7754 - val_loss: 2.2326 - val_precision: 0.4770 - val_recall: 0.20
91
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 34s 5s/step - accuracy: 0.7044 - auc: 0.9524 - loss: 1.0085 - precision: 0.8508 - recal
l: 0.4959 - val_accuracy: 0.4257 - val_auc: 0.8066 - val_loss: 2.0428 - val_precision: 0.5694 - val_recall: 0.29
97
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 33s 5s/step - accuracy: 0.7876 - auc: 0.9773 - loss: 0.7180 - precision: 0.9011 - recal
l: 0.6416 - val_accuracy: 0.4408 - val_auc: 0.8011 - val_loss: 2.0789 - val_precision: 0.5631 - val_recall: 0.29
22
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.8431 - auc: 0.9899 - loss: 0.5213 - precision: 0.9326 - recal
l: 0.7390 - val_accuracy: 0.4332 - val_auc: 0.8012 - val_loss: 2.1565 - val_precision: 0.5402 - val_recall: 0.30
48
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.9085 - auc: 0.9956 - loss: 0.3503 - precision: 0.9638 - recal
l: 0.8365 - val_accuracy: 0.4559 - val_auc: 0.7993 - val_loss: 2.3629 - val_precision: 0.5349 - val_recall: 0.34
76
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9285 - auc: 0.9976 - loss: 0.2582 - precision: 0.9752 - recal
l: 0.8818 - val_accuracy: 0.4307 - val_auc: 0.8021 - val_loss: 2.4970 - val_precision: 0.5068 - val_recall: 0.37
78
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 32s 5s/step - accuracy: 0.9570 - auc: 0.9992 - loss: 0.1659 - precision: 0.9755 - recal
l: 0.9340 - val_accuracy: 0.4408 - val_auc: 0.8051 - val_loss: 2.5254 - val_precision: 0.4765 - val_recall: 0.35
77
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9673 - auc: 0.9994 - loss: 0.1530 - precision: 0.9851 - recal
l: 0.9394 - val_accuracy: 0.4257 - val_auc: 0.7952 - val_loss: 2.8406 - val_precision: 0.4763 - val_recall: 0.38
04
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9562 - auc: 0.9993 - loss: 0.1310 - precision: 0.9726 - recal
l: 0.9422 - val_accuracy: 0.4710 - val_auc: 0.8041 - val_loss: 2.5447 - val_precision: 0.5288 - val_recall: 0.39
29
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9813 - auc: 0.9996 - loss: 0.0987 - precision: 0.9884 - recal
l: 0.9640 - val_accuracy: 0.4332 - val_auc: 0.8000 - val_loss: 2.6519 - val_precision: 0.4901 - val_recall: 0.37
53
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9797 - auc: 0.9997 - loss: 0.0756 - precision: 0.9891 - recal
l: 0.9743 - val_accuracy: 0.4257 - val_auc: 0.7952 - val_loss: 2.9259 - val_precision: 0.4985 - val_recall: 0.41
06
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9854 - auc: 0.9999 - loss: 0.0571 - precision: 0.9896 - recal
l: 0.9803 - val_accuracy: 0.4332 - val_auc: 0.7920 - val_loss: 2.7538 - val_precision: 0.4903 - val_recall: 0.38
29

Training time: 641.04 seconds
```

In [10]:
```
print("\nValidation Accuracy Model1:")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history1.history[m][-1]:.4f}")
```

```
Validation Accuracy Model1:
val_accuracy: 0.4332
val_precision: 0.4903
val_recall: 0.3829
val_auc: 0.7920
```

- Model 1 has validation accuracy of (43.32%) and a strong AUC of 0.7844. These results indicate effective overall classification across multiple bird species.

In [11]:
```
# plotting the results
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history1.history['accuracy'], label='Train Accuracy')
plt.plot(history1.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history1.history['loss'], label='Train Loss')
plt.plot(history1.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

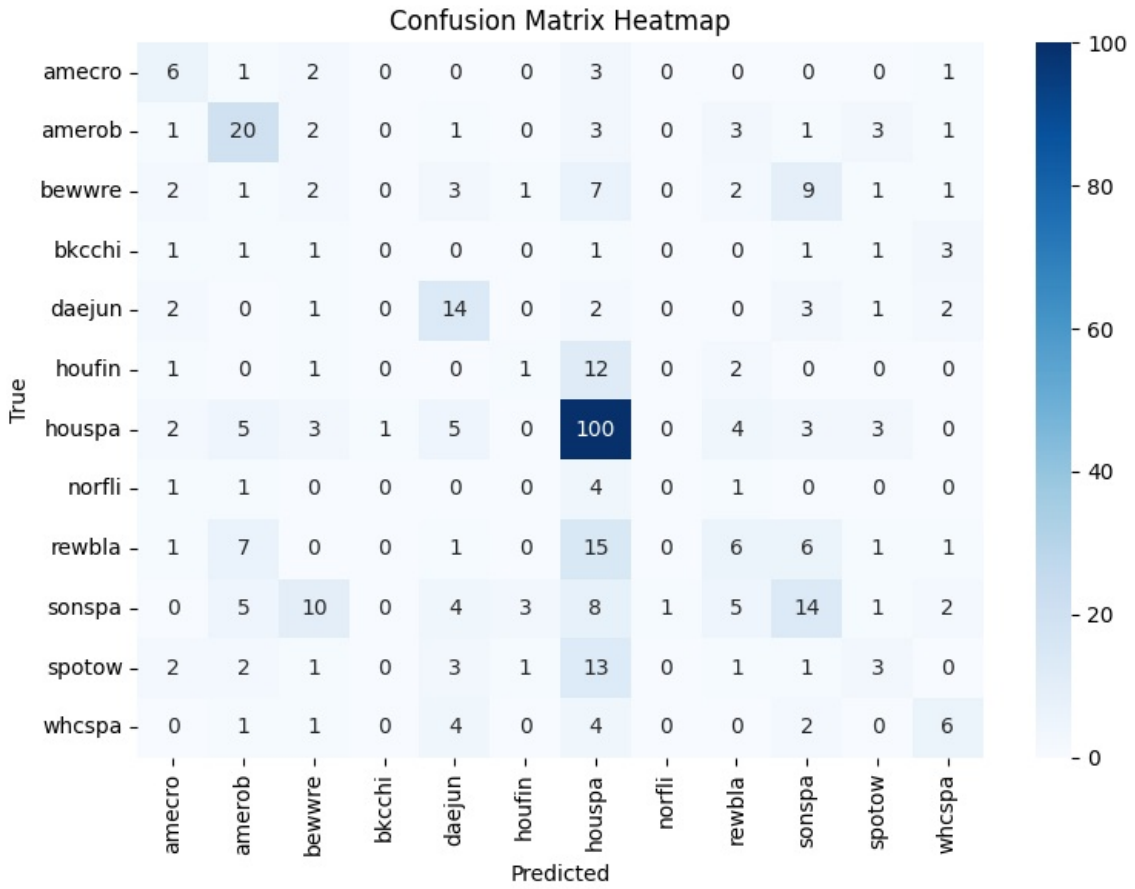


- The graphs show that the model's training accuracy keeps improving, but the validation accuracy stays around 45%, which means the model might be overfitting.
- The loss graph also shows training loss goes down , while validation loss remains mostly flat indicatingthat the model is learning the training data well but not generalizing to new data.

In [12]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

y_pred = model1_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm1 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
```

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

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 117ms/step
```



- This confusion matrix shows that the model performs best on houspa (House Sparrow), predicting it correctly 106 times. However, it struggles to distinguish between several other bird species like bewwre, rewbla, and sonspa, often confusing them with each other.

Model2

In [13]:
```
# Defining model2
np.random.seed(42)
tf.random.set_seed(42)


model2_multi = Sequential()
model2_multi.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model2_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2_multi.add(Conv2D(32, (3, 3), activation='relu'))
model2_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2_multi.add(Flatten())
model2_multi.add(Dense(64, activation='relu'))
model2_multi.add(Dropout(0.3))

model2_multi.add(Dense(12, activation='softmax'))

model2_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [14]:
```
# Tracking the model training time
import time
start_time = time.time()
```

```
history2 = model2_multi.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=15,
    batch_size=128
)

end_time = time.time()
time2 = end_time - start_time
print(f"\nTraining time: {time2:.2f} seconds")
```

```
Epoch 1/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 11s 694ms/step - accuracy: 0.1675 - auc: 0.5497 - loss: 225.6911 - precision: 0.1647
- recall: 0.1556 - val_accuracy: 0.3174 - val_auc: 0.6720 - val_loss: 2.3636 - val_precision: 0.0000e+00 - val_r
ecall: 0.0000e+00
Epoch 2/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 654ms/step - accuracy: 0.1788 - auc: 0.6281 - loss: 2.4847 - precision: 0.0977 - r
ecall: 0.0025 - val_accuracy: 0.2771 - val_auc: 0.6823 - val_loss: 2.3905 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 3/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 670ms/step - accuracy: 0.2556 - auc: 0.6828 - loss: 2.4334 - precision: 0.3094 - r
ecall: 0.0339 - val_accuracy: 0.3149 - val_auc: 0.7282 - val_loss: 2.2707 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 4/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 681ms/step - accuracy: 0.3738 - auc: 0.7624 - loss: 2.1663 - precision: 0.5728 - r
ecall: 0.0493 - val_accuracy: 0.3778 - val_auc: 0.7536 - val_loss: 2.1355 - val_precision: 0.3929 - val_recall:
0.0554
Epoch 5/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 676ms/step - accuracy: 0.5174 - auc: 0.8676 - loss: 1.7304 - precision: 0.6833 - r
ecall: 0.1256 - val_accuracy: 0.4081 - val_auc: 0.7968 - val_loss: 1.9734 - val_precision: 0.5104 - val_recall:
0.1234
Epoch 6/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 668ms/step - accuracy: 0.6408 - auc: 0.9278 - loss: 1.2562 - precision: 0.8154 - r
ecall: 0.3680 - val_accuracy: 0.4131 - val_auc: 0.7979 - val_loss: 2.0009 - val_precision: 0.5152 - val_recall:
0.1285
Epoch 7/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 661ms/step - accuracy: 0.7841 - auc: 0.9761 - loss: 0.7922 - precision: 0.9134 - r
ecall: 0.5615 - val_accuracy: 0.4307 - val_auc: 0.7871 - val_loss: 2.3146 - val_precision: 0.4957 - val_recall:
0.2897
Epoch 8/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 653ms/step - accuracy: 0.8689 - auc: 0.9894 - loss: 0.4926 - precision: 0.9371 - r
ecall: 0.7634 - val_accuracy: 0.4358 - val_auc: 0.8013 - val_loss: 2.2728 - val_precision: 0.4791 - val_recall:
0.3174
Epoch 9/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 661ms/step - accuracy: 0.9118 - auc: 0.9940 - loss: 0.3379 - precision: 0.9584 - r
ecall: 0.8525 - val_accuracy: 0.4282 - val_auc: 0.7932 - val_loss: 2.2156 - val_precision: 0.5138 - val_recall:
0.2821
Epoch 10/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 671ms/step - accuracy: 0.9505 - auc: 0.9960 - loss: 0.2284 - precision: 0.9681 - r
ecall: 0.9132 - val_accuracy: 0.4005 - val_auc: 0.7658 - val_loss: 2.6874 - val_precision: 0.4648 - val_recall:
0.2997
Epoch 11/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 693ms/step - accuracy: 0.9315 - auc: 0.9961 - loss: 0.3092 - precision: 0.9578 - r
ecall: 0.8611 - val_accuracy: 0.4207 - val_auc: 0.7729 - val_loss: 2.5517 - val_precision: 0.4835 - val_recall:
0.3325
Epoch 12/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 686ms/step - accuracy: 0.9635 - auc: 0.9990 - loss: 0.1784 - precision: 0.9795 - r
ecall: 0.9316 - val_accuracy: 0.4106 - val_auc: 0.7864 - val_loss: 2.6806 - val_precision: 0.4804 - val_recall:
0.3401
Epoch 13/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 687ms/step - accuracy: 0.9716 - auc: 0.9991 - loss: 0.1314 - precision: 0.9805 - r
ecall: 0.9521 - val_accuracy: 0.3980 - val_auc: 0.7801 - val_loss: 2.9548 - val_precision: 0.4441 - val_recall:
0.3300
Epoch 14/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 671ms/step - accuracy: 0.9817 - auc: 0.9995 - loss: 0.0800 - precision: 0.9859 - r
ecall: 0.9755 - val_accuracy: 0.4257 - val_auc: 0.7916 - val_loss: 2.8339 - val_precision: 0.5017 - val_recall:
0.3728
Epoch 15/15
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 668ms/step - accuracy: 0.9873 - auc: 0.9997 - loss: 0.0545 - precision: 0.9915 - r
ecall: 0.9847 - val_accuracy: 0.4635 - val_auc: 0.7810 - val_loss: 3.0003 - val_precision: 0.5081 - val_recall:
0.3929

Training time: 134.07 seconds
```

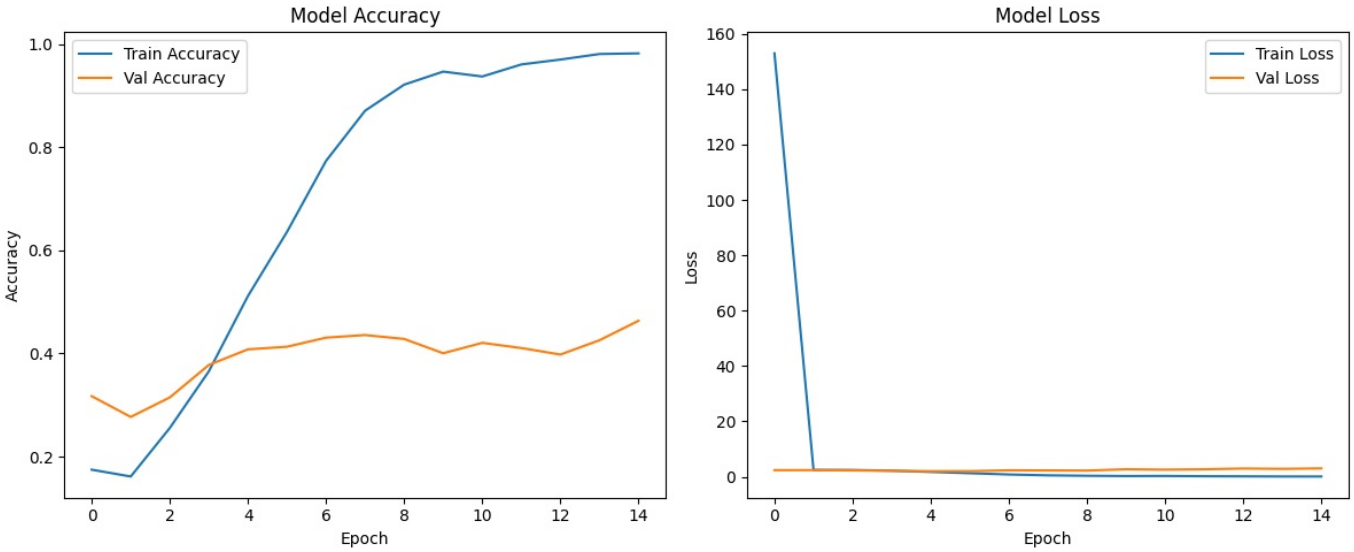
In [15]:

```
print("\nValidation Accuracy Model2:")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history2.history[m][-1]:.4f}")
```

```
Validation Accuracy Model2:
val_accuracy: 0.4635
val_precision: 0.5081
val_recall: 0.3929
val_auc: 0.7810
```

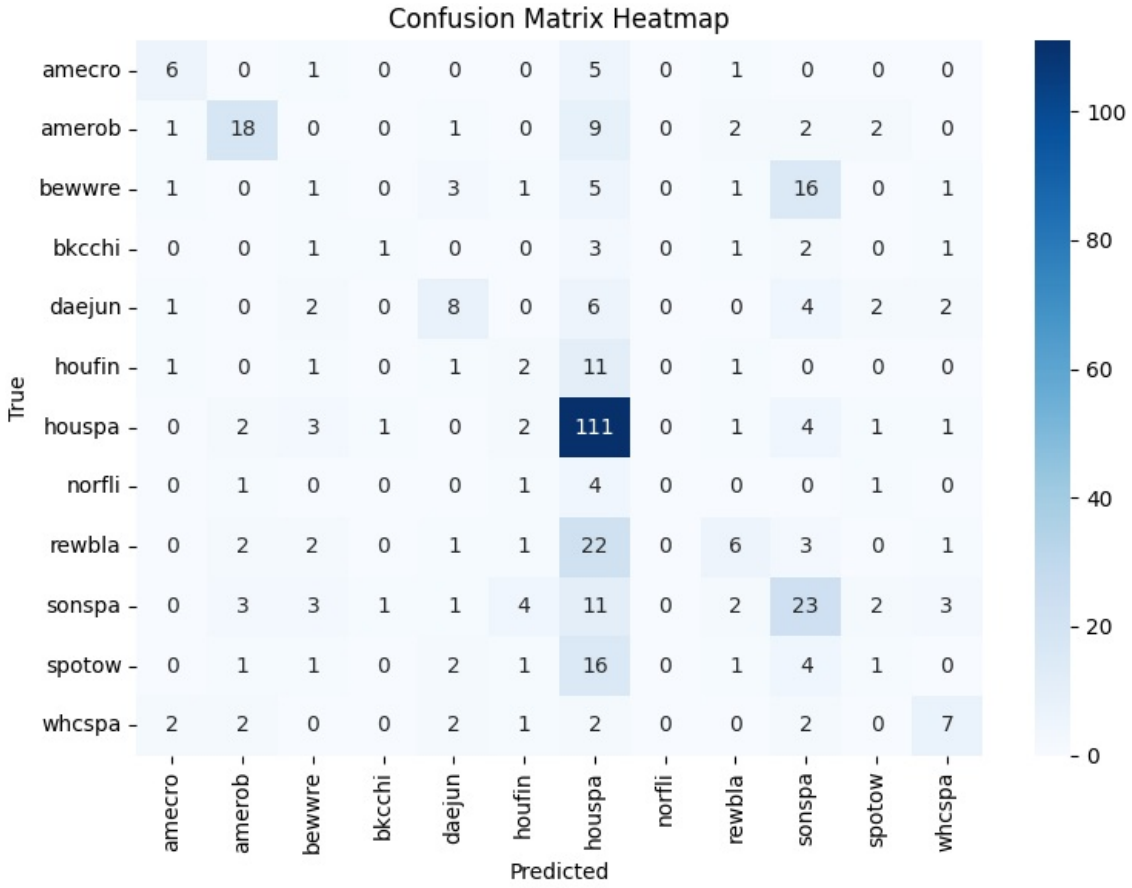
- Model 2 has validation accuracy of (46.35%) and a strong AUC of 0.7810. These results shows that it has 3% moere accuracy than model1.

In [16]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history2.history['accuracy'], label='Train Accuracy')
plt.plot(history2.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history2.history['loss'], label='Train Loss')
plt.plot(history2.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



In [17]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

y_pred = model2_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm2 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step
```



- The model predicts House Sparrow and song sparrow very accurately, with minimal confusion.

Model1-A

In [18]:

```
np.random.seed(123)
tf.random.set_seed(123)

model1a_multi = Sequential()
model1a_multi.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model1a_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1a_multi.add(Conv2D(64, (3, 3), activation='relu'))
model1a_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1a_multi.add(Flatten())
model1a_multi.add(Dense(128, activation='relu'))
model1a_multi.add(Dropout(0.4))

model1a_multi.add(Dense(12, activation='softmax'))

model1a_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [19]:

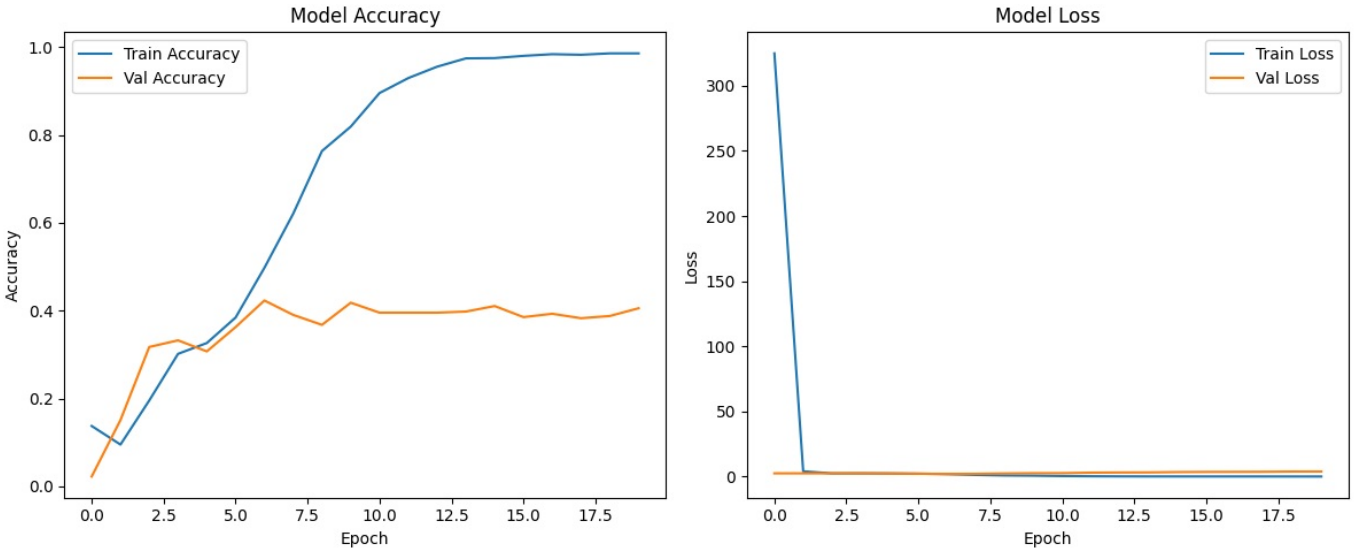
```python
start_time = time.time()
history3 = model1a_multi.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=128
)

end_time = time.time()
time3 = end_time - start_time
print(f"\nTraining time: {time3:.2f} seconds")
```

```
Epoch 1/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 2s/step - accuracy: 0.1474 - auc: 0.5388 - loss: 490.2636 - precision: 0.1479 - r
ecall: 0.1436 - val_accuracy: 0.0227 - val_auc: 0.5114 - val_loss: 2.4803 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 2/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.1077 - auc: 0.5389 - loss: 4.6898 - precision: 0.1610 - rec
all: 0.0580 - val_accuracy: 0.1511 - val_auc: 0.5459 - val_loss: 2.4739 - val_precision: 0.0000e+00 - val_recall
: 0.0000e+00
Epoch 3/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.1850 - auc: 0.5440 - loss: 2.4723 - precision: 0.0000e+00 -
recall: 0.0000e+00 - val_accuracy: 0.3174 - val_auc: 0.5255 - val_loss: 2.4738 - val_precision: 0.0000e+00 - val
_recall: 0.0000e+00
Epoch 4/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.3251 - auc: 0.6218 - loss: 2.4475 - precision: 0.2931 - rec
all: 0.0156 - val_accuracy: 0.3325 - val_auc: 0.6755 - val_loss: 2.4626 - val_precision: 0.0000e+00 - val_recall
: 0.0000e+00
Epoch 5/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.3568 - auc: 0.7039 - loss: 2.4295 - precision: 0.2080 - rec
all: 0.0055 - val_accuracy: 0.3073 - val_auc: 0.7320 - val_loss: 2.3537 - val_precision: 0.8000 - val_recall: 0.
0101
Epoch 6/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.3726 - auc: 0.7713 - loss: 2.2719 - precision: 0.9558 - rec
all: 0.0228 - val_accuracy: 0.3627 - val_auc: 0.7614 - val_loss: 2.1646 - val_precision: 0.2973 - val_recall: 0.
0277
Epoch 7/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 32s 2s/step - accuracy: 0.4831 - auc: 0.8300 - loss: 1.8891 - precision: 0.6481 - rec
all: 0.1299 - val_accuracy: 0.4232 - val_auc: 0.7944 - val_loss: 2.0609 - val_precision: 0.5044 - val_recall: 0.
1436
Epoch 8/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 32s 2s/step - accuracy: 0.5993 - auc: 0.9144 - loss: 1.3373 - precision: 0.7948 - rec
all: 0.3693 - val_accuracy: 0.3904 - val_auc: 0.7869 - val_loss: 2.1221 - val_precision: 0.4809 - val_recall: 0.
1587
Epoch 9/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 32s 2s/step - accuracy: 0.7471 - auc: 0.9612 - loss: 0.9261 - precision: 0.8851 - rec
all: 0.5404 - val_accuracy: 0.3678 - val_auc: 0.7654 - val_loss: 2.3917 - val_precision: 0.4556 - val_recall: 0.
2065
Epoch 10/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.8065 - auc: 0.9766 - loss: 0.7406 - precision: 0.9126 - rec
all: 0.6287 - val_accuracy: 0.4181 - val_auc: 0.7855 - val_loss: 2.6065 - val_precision: 0.4743 - val_recall: 0.
3023
Epoch 11/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.8809 - auc: 0.9901 - loss: 0.4354 - precision: 0.9246 - rec
all: 0.7986 - val_accuracy: 0.3955 - val_auc: 0.7822 - val_loss: 2.6364 - val_precision: 0.4563 - val_recall: 0.
2897
Epoch 12/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.9286 - auc: 0.9970 - loss: 0.2632 - precision: 0.9641 - rec
all: 0.8865 - val_accuracy: 0.3955 - val_auc: 0.7760 - val_loss: 2.9961 - val_precision: 0.4621 - val_recall: 0.
3375
Epoch 13/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 31s 2s/step - accuracy: 0.9571 - auc: 0.9989 - loss: 0.1614 - precision: 0.9749 - rec
all: 0.9297 - val_accuracy: 0.3955 - val_auc: 0.7766 - val_loss: 3.1419 - val_precision: 0.4400 - val_recall: 0.
3325
Epoch 14/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 32s 2s/step - accuracy: 0.9781 - auc: 0.9994 - loss: 0.1087 - precision: 0.9875 - rec
all: 0.9576 - val_accuracy: 0.3980 - val_auc: 0.7760 - val_loss: 3.2108 - val_precision: 0.4295 - val_recall: 0.
3375
Epoch 15/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.9758 - auc: 0.9994 - loss: 0.0937 - precision: 0.9864 - rec
all: 0.9628 - val_accuracy: 0.4106 - val_auc: 0.7697 - val_loss: 3.4532 - val_precision: 0.4303 - val_recall: 0.
3501
Epoch 16/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 31s 2s/step - accuracy: 0.9811 - auc: 0.9996 - loss: 0.0762 - precision: 0.9842 - rec
all: 0.9718 - val_accuracy: 0.3854 - val_auc: 0.7689 - val_loss: 3.5760 - val_precision: 0.4149 - val_recall: 0.
3375
Epoch 17/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 32s 3s/step - accuracy: 0.9813 - auc: 0.9991 - loss: 0.0869 - precision: 0.9856 - rec
all: 0.9717 - val_accuracy: 0.3929 - val_auc: 0.7609 - val_loss: 3.6187 - val_precision: 0.4182 - val_recall: 0.
3350
Epoch 18/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.9829 - auc: 0.9993 - loss: 0.0803 - precision: 0.9859 - rec
all: 0.9692 - val_accuracy: 0.3829 - val_auc: 0.7631 - val_loss: 3.6721 - val_precision: 0.4322 - val_recall: 0.
3451
Epoch 19/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.9850 - auc: 0.9996 - loss: 0.0590 - precision: 0.9900 - rec
all: 0.9770 - val_accuracy: 0.3879 - val_auc: 0.7549 - val_loss: 3.8591 - val_precision: 0.4218 - val_recall: 0.
3602
Epoch 20/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 33s 3s/step - accuracy: 0.9867 - auc: 0.9999 - loss: 0.0456 - precision: 0.9921 - rec
all: 0.9839 - val_accuracy: 0.4055 - val_auc: 0.7673 - val_loss: 3.8660 - val_precision: 0.4332 - val_recall: 0.
3678

Training time: 659.30 seconds
```

In [20]:
```
print("\nValidation Accuracy Model3:")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history3.history[m][-1]:.4f}")
```

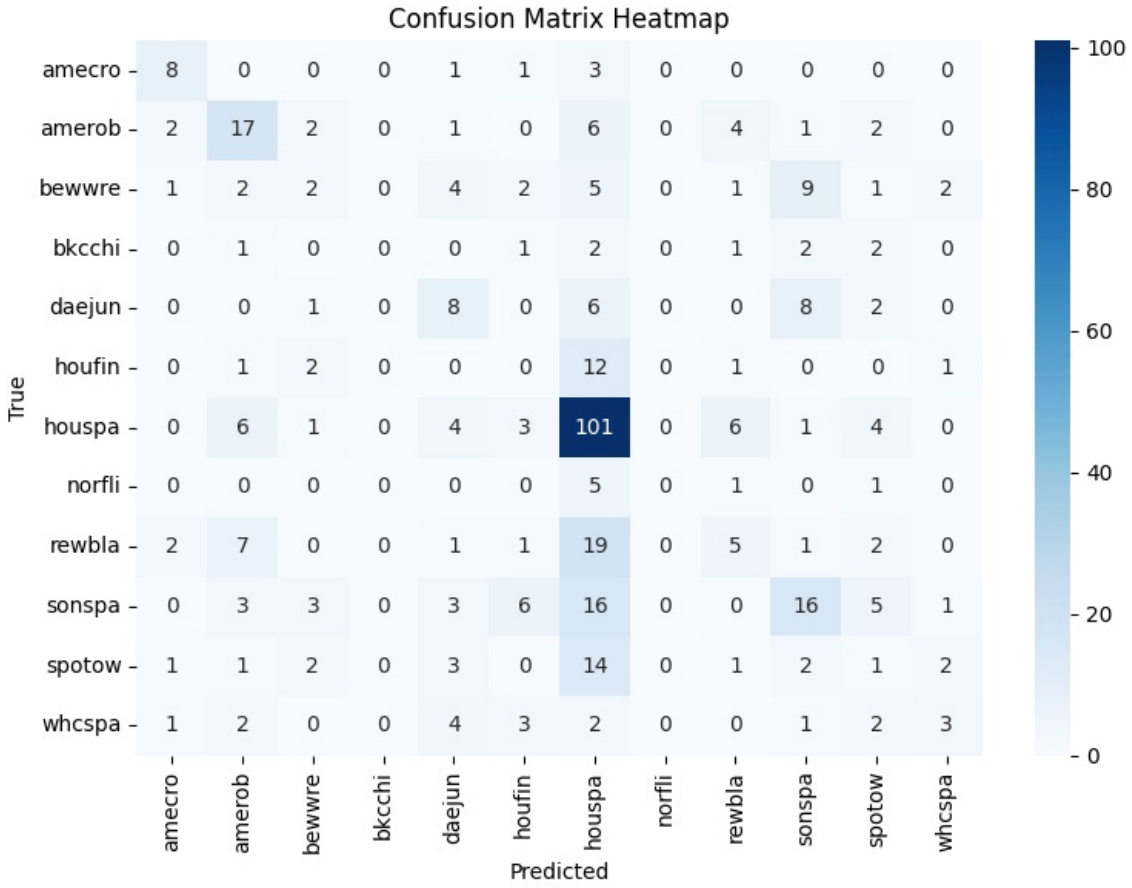
```
Validation Accuracy Model3:
val_accuracy: 0.4055
val_precision: 0.4332
val_recall: 0.3678
val_auc: 0.7673
```

In [21]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history3.history['accuracy'], label='Train Accuracy')
plt.plot(history3.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history3.history['loss'], label='Train Loss')
plt.plot(history3.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



In [22]:
```
y_pred = model1a_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm3 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm3, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 125ms/step
```



Model1-B

In [23]:
```
np.random.seed(160)
tf.random.set_seed(160)

model1b_multi = Sequential()
model1b_multi.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model1b_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1b_multi.add(Conv2D(64, (3, 3), activation='relu'))
model1b_multi.add(MaxPooling2D(pool_size=(2, 2)))

model1b_multi.add(Flatten())
model1b_multi.add(Dense(128, activation='relu'))
model1b_multi.add(Dropout(0.3))

model1b_multi.add(Dense(12, activation='softmax'))

model1b_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [24]:
```
start_time = time.time()
```

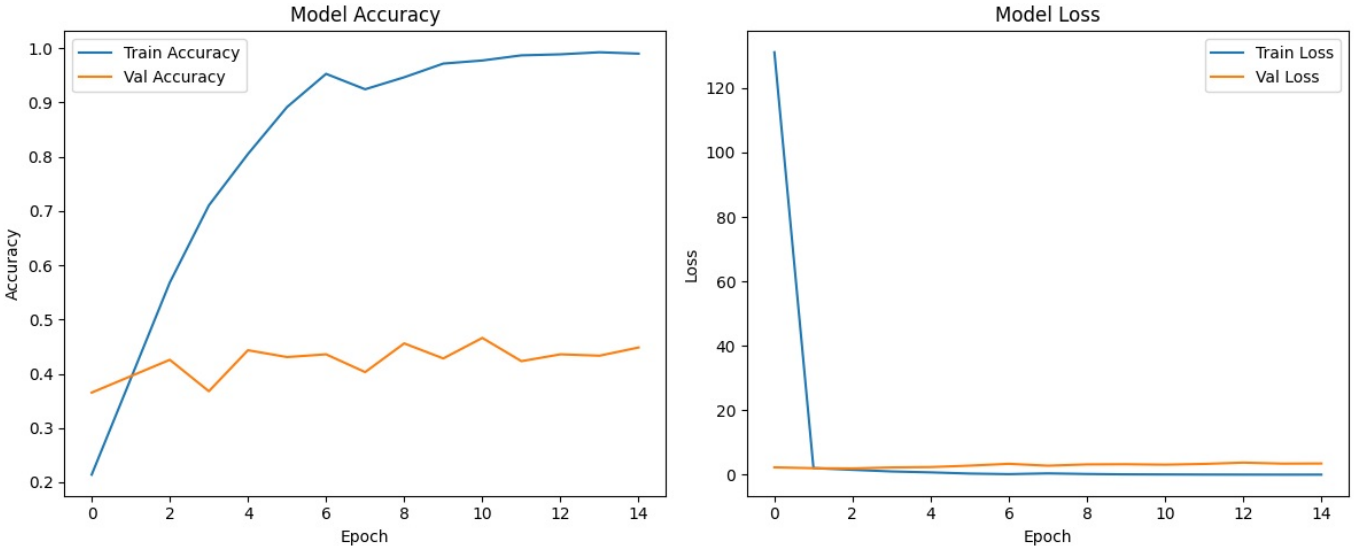
```
history4= model1b_multi.fit(
    X_train, y_train,
    epochs=15,
    batch_size=64,
    validation_data=(X_test, y_test)
)

end_time = time.time()
time4 = end_time - start_time

print(f"Training time: {time4:.2f} seconds ({time4 / 60:.2f} minutes)")
```

```
Epoch 1/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 37s 1s/step - accuracy: 0.1688 - auc: 0.5577 - loss: 277.2318 - precision: 0.1606 - r
ecall: 0.1203 - val_accuracy: 0.3652 - val_auc: 0.7518 - val_loss: 2.2955 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 34s 1s/step - accuracy: 0.3851 - auc: 0.7703 - loss: 2.1832 - precision: 0.5867 - rec
all: 0.0304 - val_accuracy: 0.3955 - val_auc: 0.7954 - val_loss: 2.0291 - val_precision: 0.5455 - val_recall: 0.
0453
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.5282 - auc: 0.8698 - loss: 1.6901 - precision: 0.7639 - rec
all: 0.1532 - val_accuracy: 0.4257 - val_auc: 0.8044 - val_loss: 1.9602 - val_precision: 0.5440 - val_recall: 0.
1713
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.6821 - auc: 0.9448 - loss: 1.1306 - precision: 0.8553 - rec
all: 0.4145 - val_accuracy: 0.3678 - val_auc: 0.8055 - val_loss: 2.2662 - val_precision: 0.4735 - val_recall: 0.
2695
Epoch 5/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.7801 - auc: 0.9670 - loss: 0.8575 - precision: 0.8845 - rec
all: 0.5553 - val_accuracy: 0.4433 - val_auc: 0.8014 - val_loss: 2.3811 - val_precision: 0.5075 - val_recall: 0.
3401
Epoch 6/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.8629 - auc: 0.9911 - loss: 0.4389 - precision: 0.9223 - rec
all: 0.7953 - val_accuracy: 0.4307 - val_auc: 0.7871 - val_loss: 2.8283 - val_precision: 0.4704 - val_recall: 0.
3602
Epoch 7/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 35s 1s/step - accuracy: 0.9368 - auc: 0.9979 - loss: 0.2201 - precision: 0.9600 - rec
all: 0.8997 - val_accuracy: 0.4358 - val_auc: 0.7741 - val_loss: 3.3975 - val_precision: 0.4688 - val_recall: 0.
3980
Epoch 8/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 35s 1s/step - accuracy: 0.9330 - auc: 0.9913 - loss: 0.3123 - precision: 0.9470 - rec
all: 0.9139 - val_accuracy: 0.4030 - val_auc: 0.7788 - val_loss: 2.8058 - val_precision: 0.4586 - val_recall: 0.
3350
Epoch 9/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9369 - auc: 0.9922 - loss: 0.2830 - precision: 0.9527 - rec
all: 0.8920 - val_accuracy: 0.4559 - val_auc: 0.7755 - val_loss: 3.2140 - val_precision: 0.5061 - val_recall: 0.
4207
Epoch 10/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9678 - auc: 0.9993 - loss: 0.1325 - precision: 0.9752 - rec
all: 0.9502 - val_accuracy: 0.4282 - val_auc: 0.7860 - val_loss: 3.2643 - val_precision: 0.4679 - val_recall: 0.
3854
Epoch 11/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9704 - auc: 0.9995 - loss: 0.1152 - precision: 0.9789 - rec
all: 0.9520 - val_accuracy: 0.4660 - val_auc: 0.7932 - val_loss: 3.1306 - val_precision: 0.4882 - val_recall: 0.
4156
Epoch 12/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9834 - auc: 0.9991 - loss: 0.0644 - precision: 0.9870 - rec
all: 0.9752 - val_accuracy: 0.4232 - val_auc: 0.7987 - val_loss: 3.3640 - val_precision: 0.4599 - val_recall: 0.
3904
Epoch 13/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 34s 1s/step - accuracy: 0.9837 - auc: 0.9998 - loss: 0.0568 - precision: 0.9879 - rec
all: 0.9802 - val_accuracy: 0.4358 - val_auc: 0.7812 - val_loss: 3.7600 - val_precision: 0.4718 - val_recall: 0.
4207
Epoch 14/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9933 - auc: 1.0000 - loss: 0.0276 - precision: 0.9953 - rec
all: 0.9916 - val_accuracy: 0.4332 - val_auc: 0.7918 - val_loss: 3.4574 - val_precision: 0.4687 - val_recall: 0.
3955
Epoch 15/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9900 - auc: 0.9999 - loss: 0.0329 - precision: 0.9932 - rec
all: 0.9883 - val_accuracy: 0.4484 - val_auc: 0.7943 - val_loss: 3.4947 - val_precision: 0.4926 - val_recall: 0.
4207
Training time: 506.91 seconds (8.45 minutes)
```

In [25]:
```
print("\nValidation Accuracy Model4:")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history4.history[m][-1]:.4f}")
```

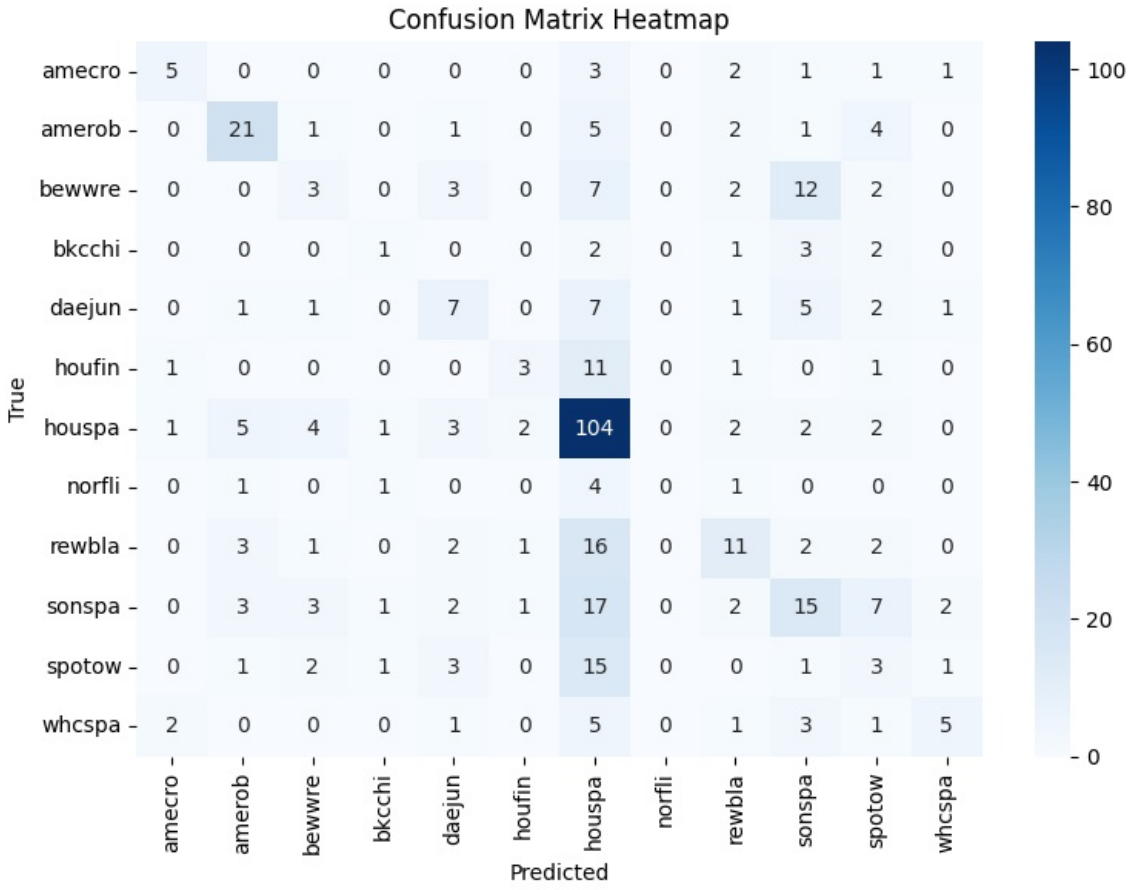
```
Validation Accuracy Model4:
val_accuracy: 0.4484
val_precision: 0.4926
val_recall: 0.4207
val_auc: 0.7943
```

In [26]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history4.history['accuracy'], label='Train Accuracy')
plt.plot(history4.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history4.history['loss'], label='Train Loss')
plt.plot(history4.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



In [27]:
```
y_pred = model1b_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm4 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm4, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

13/13 ━━━━━━━━━━━━━━━━━━━━ **2s** 121ms/step



Model2-A

In [28]:

```
np.random.seed(160)
tf.random.set_seed(160)


model2a_multi = Sequential()
model2a_multi.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model2a_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2a_multi.add(Conv2D(32, (3, 3), activation='relu'))
model2a_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2a_multi.add(Flatten())
model2a_multi.add(Dense(64, activation='relu'))
model2a_multi.add(Dropout(0.4))

model2a_multi.add(Dense(12, activation='softmax'))

model2a_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
```

```
        Recall(name='recall'),
        AUC(name='auc')
    ]
)

model2a_multi.summary()
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 126, 515, 16)	160
max_pooling2d_8 (MaxPooling2D)	(None, 63, 257, 16)	0
conv2d_9 (Conv2D)	(None, 61, 255, 32)	4,640
max_pooling2d_9 (MaxPooling2D)	(None, 30, 127, 32)	0
flatten_4 (Flatten)	(None, 121920)	0
dense_8 (Dense)	(None, 64)	7,802,944
dropout_4 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 12)	780

Total params: 7,808,524 (29.79 MB)

Trainable params: 7,808,524 (29.79 MB)

Non-trainable params: 0 (0.00 B)

In [29]:
```
start_time = time.time()
history6= model2a_multi.fit(
    X_train, y_train,
    epochs=15,
    batch_size=32,
    validation_data=(X_test, y_test)
)

end_time = time.time()
time6 = end_time - start_time

print(f"Training time: {time6:.2f} seconds ({time6 / 60:.2f} minutes)")
```

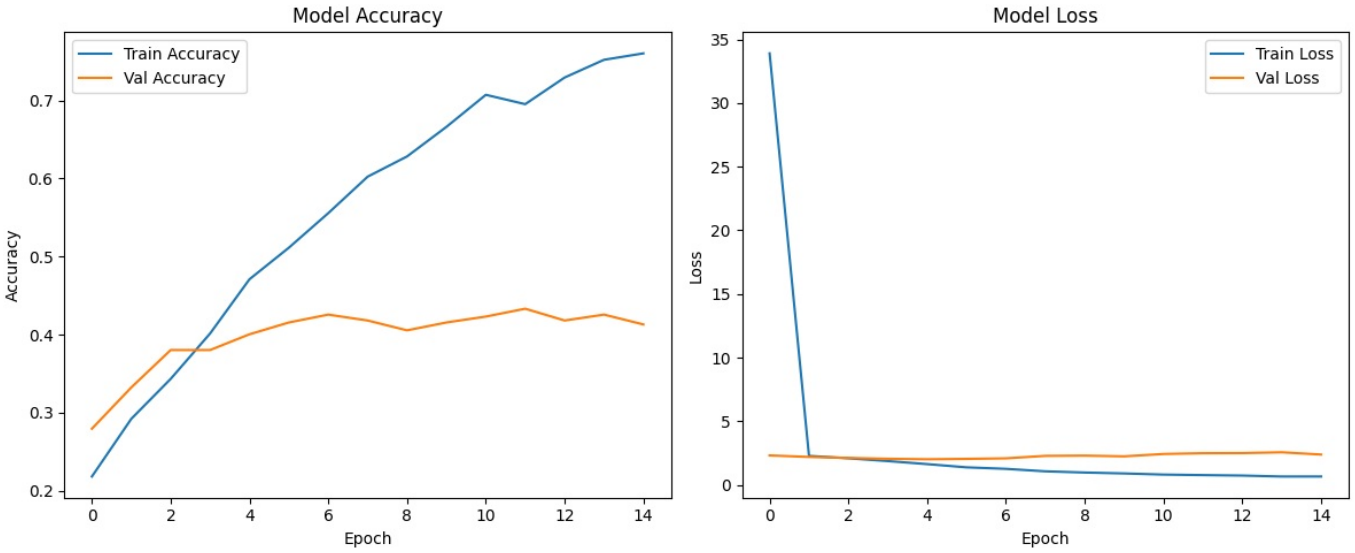
```
Epoch 1/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 218ms/step - accuracy: 0.1576 - auc: 0.5262 - loss: 89.9453 - precision: 0.1160 -
recall: 0.0574 - val_accuracy: 0.2796 - val_auc: 0.6792 - val_loss: 2.3146 - val_precision: 0.0000e+00 - val_rec
all: 0.0000e+00
Epoch 2/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 211ms/step - accuracy: 0.2761 - auc: 0.6786 - loss: 2.3032 - precision: 0.6057 -
recall: 0.0140 - val_accuracy: 0.3325 - val_auc: 0.7132 - val_loss: 2.1947 - val_precision: 0.5946 - val_recall:
0.0554
Epoch 3/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 210ms/step - accuracy: 0.3382 - auc: 0.7604 - loss: 2.0945 - precision: 0.7583 -
recall: 0.0832 - val_accuracy: 0.3804 - val_auc: 0.7514 - val_loss: 2.1184 - val_precision: 0.7500 - val_recall:
0.0378
Epoch 4/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 213ms/step - accuracy: 0.4139 - auc: 0.8156 - loss: 1.8752 - precision: 0.7625 -
recall: 0.1354 - val_accuracy: 0.3804 - val_auc: 0.7733 - val_loss: 2.0503 - val_precision: 0.6207 - val_recall:
0.0453
Epoch 5/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 213ms/step - accuracy: 0.4658 - auc: 0.8616 - loss: 1.6769 - precision: 0.8261 -
recall: 0.2101 - val_accuracy: 0.4005 - val_auc: 0.7823 - val_loss: 2.0135 - val_precision: 0.6559 - val_recall:
0.1537
Epoch 6/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 213ms/step - accuracy: 0.4949 - auc: 0.9019 - loss: 1.4338 - precision: 0.8300 -
recall: 0.3012 - val_accuracy: 0.4156 - val_auc: 0.7857 - val_loss: 2.0423 - val_precision: 0.6354 - val_recall:
0.1537
Epoch 7/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 212ms/step - accuracy: 0.5460 - auc: 0.9238 - loss: 1.2751 - precision: 0.8316 -
recall: 0.3572 - val_accuracy: 0.4257 - val_auc: 0.7920 - val_loss: 2.0858 - val_precision: 0.6071 - val_recall:
0.2569
Epoch 8/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 211ms/step - accuracy: 0.5891 - auc: 0.9471 - loss: 1.0776 - precision: 0.8539 -
recall: 0.4484 - val_accuracy: 0.4181 - val_auc: 0.7821 - val_loss: 2.2802 - val_precision: 0.5674 - val_recall:
0.3073
Epoch 9/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 216ms/step - accuracy: 0.6204 - auc: 0.9566 - loss: 0.9760 - precision: 0.8708 -
recall: 0.4972 - val_accuracy: 0.4055 - val_auc: 0.7879 - val_loss: 2.2989 - val_precision: 0.5388 - val_recall:
0.2972
Epoch 10/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 214ms/step - accuracy: 0.6574 - auc: 0.9627 - loss: 0.8983 - precision: 0.8973 -
recall: 0.5350 - val_accuracy: 0.4156 - val_auc: 0.7896 - val_loss: 2.2420 - val_precision: 0.5803 - val_recall:
0.2821
Epoch 11/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 215ms/step - accuracy: 0.6881 - auc: 0.9693 - loss: 0.8198 - precision: 0.8711 -
recall: 0.5819 - val_accuracy: 0.4232 - val_auc: 0.7922 - val_loss: 2.4287 - val_precision: 0.5569 - val_recall:
0.3451
Epoch 12/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 215ms/step - accuracy: 0.7021 - auc: 0.9742 - loss: 0.7545 - precision: 0.8710 -
recall: 0.6122 - val_accuracy: 0.4332 - val_auc: 0.7873 - val_loss: 2.4890 - val_precision: 0.5542 - val_recall:
0.3350
Epoch 13/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 214ms/step - accuracy: 0.7285 - auc: 0.9751 - loss: 0.7277 - precision: 0.8831 -
recall: 0.6360 - val_accuracy: 0.4181 - val_auc: 0.7833 - val_loss: 2.4983 - val_precision: 0.5391 - val_recall:
0.3300
Epoch 14/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 219ms/step - accuracy: 0.7612 - auc: 0.9800 - loss: 0.6484 - precision: 0.8878 -
recall: 0.6555 - val_accuracy: 0.4257 - val_auc: 0.7851 - val_loss: 2.5617 - val_precision: 0.5245 - val_recall:
0.3501
Epoch 15/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 215ms/step - accuracy: 0.7615 - auc: 0.9785 - loss: 0.6538 - precision: 0.8703 -
recall: 0.6647 - val_accuracy: 0.4131 - val_auc: 0.7976 - val_loss: 2.3840 - val_precision: 0.5370 - val_recall:
0.2922
Training time: 162.97 seconds (2.72 minutes)
```

In [30]:
```
print("\nValidation Accuracy Model6:")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history6.history[m][-1]:.4f}")
```

```
Validation Accuracy Model6:
val_accuracy: 0.4131
val_precision: 0.5370
val_recall: 0.2922
val_auc: 0.7976
```

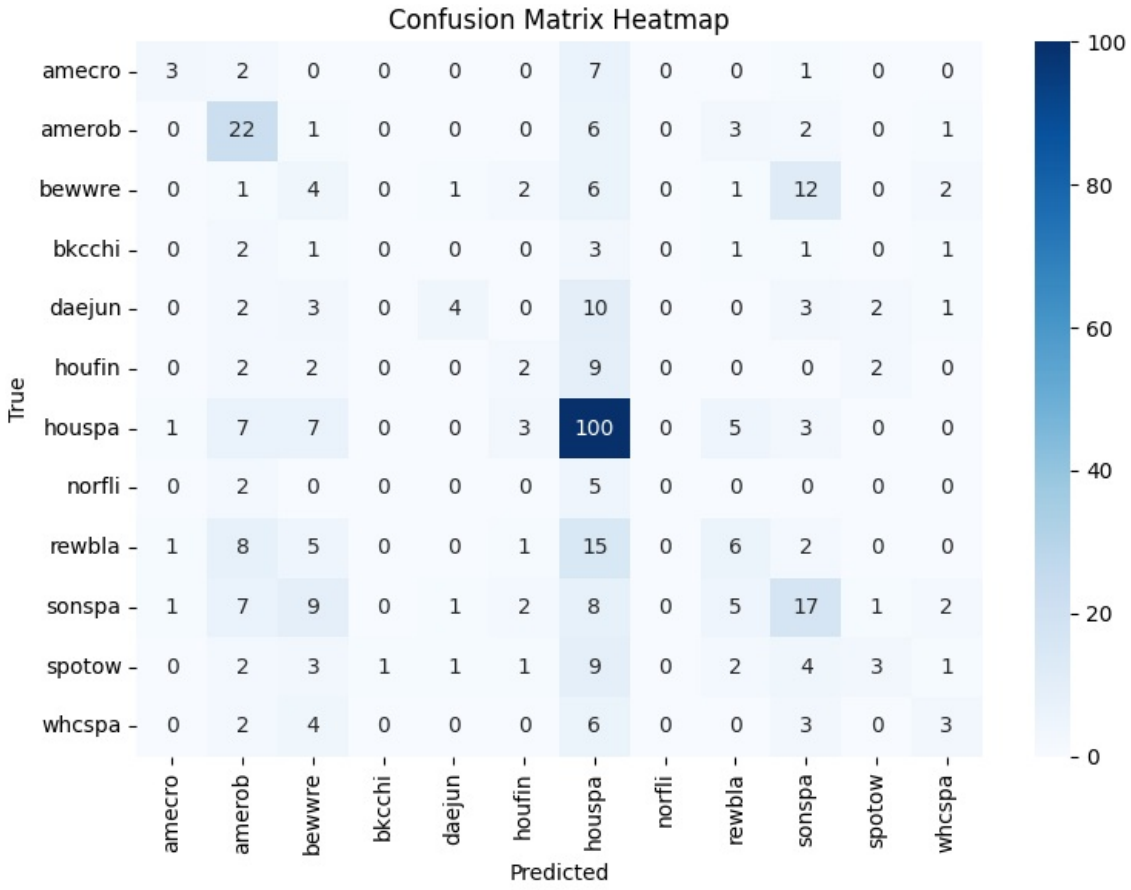
In [31]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history6.history['accuracy'], label='Train Accuracy')
plt.plot(history6.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history6.history['loss'], label='Train Loss')
```

```
plt.plot(history6.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



In [32]:
```
y_pred = model2a_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm6 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm6, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

13/13 ━━━━━━━━━━━━━━━━━━━━ **1s** 51ms/step



Addressing class imbalance issue

In [ ]:
```
* As we can see by tuning both cnn convolutional layers with different dropout rates and batches, still model i
* To address these issue of class imbalance let's use class weight for penalizing wrong prediction og lower cla
* The class weights were used on  models which achieved good accuracy in previous model to see if it affects ac
* Models used are model1b and model2 with class weights.
```

In [7]:
```
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_encoded),
    y=y_encoded
```

```
)

# Convert to dictionary
class_weights_dict = dict(enumerate(class_weights))
```

In [13]:
```
np.random.seed(160)
tf.random.set_seed(160)

model1b_weights = Sequential()
model1b_weights.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model1b_weights.add(MaxPooling2D(pool_size=(2, 2)))

model1b_weights.add(Conv2D(64, (3, 3), activation='relu'))
model1b_weights.add(MaxPooling2D(pool_size=(2, 2)))

model1b_weights.add(Flatten())
model1b_weights.add(Dense(128, activation='relu'))
model1b_weights.add(Dropout(0.3))
model1b_weights.add(Dense(12, activation='softmax'))

model1b_weights.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

In [14]:
```
import time
start_time = time.time()
history7= model1b_weights.fit(
    X_train, y_train,
    epochs=15,
    batch_size=32,
    validation_data=(X_test, y_test),
    class_weight=class_weights_dict
)

end_time = time.time()
time7 = end_time - start_time

print(f"Training time: {time7:.2f} seconds ({time7 / 60:.2f} minutes)")
```

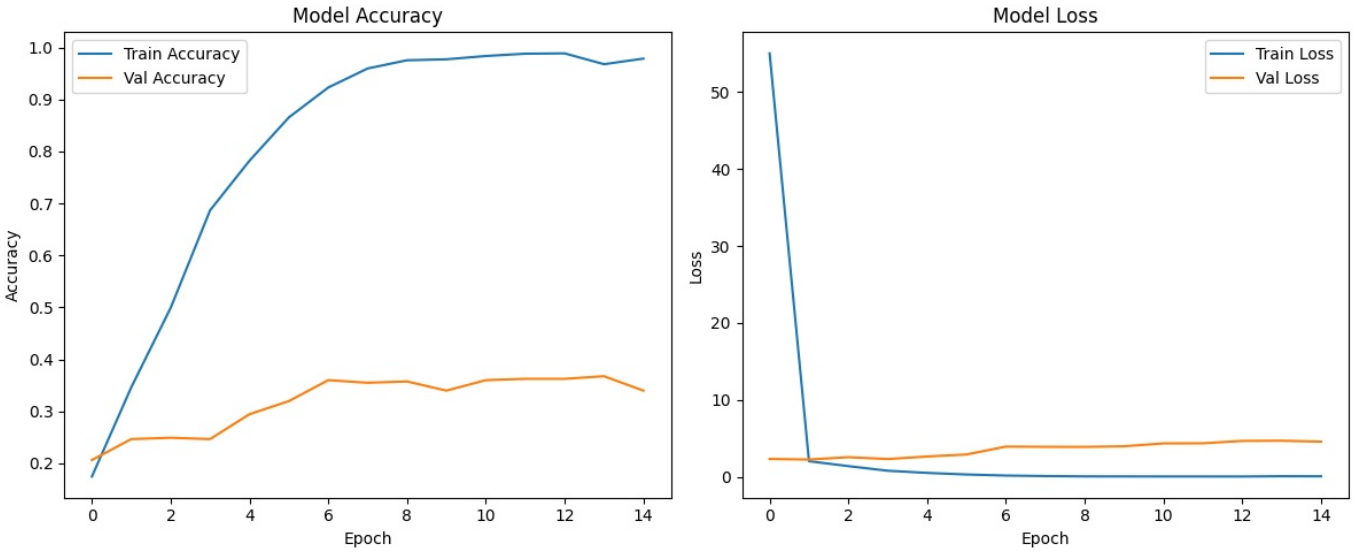
```
Epoch 1/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 772ms/step - accuracy: 0.1300 - auc: 0.5399 - loss: 153.5328 - precision: 0.0898
- recall: 0.0462 - val_accuracy: 0.2065 - val_auc: 0.6707 - val_loss: 2.3228 - val_precision: 0.5000 - val_recal
l: 0.0076
Epoch 2/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 800ms/step - accuracy: 0.3086 - auc: 0.7453 - loss: 2.2780 - precision: 0.7043 -
recall: 0.0409 - val_accuracy: 0.2469 - val_auc: 0.7207 - val_loss: 2.2517 - val_precision: 0.6034 - val_recall:
0.0882
Epoch 3/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 793ms/step - accuracy: 0.4634 - auc: 0.8686 - loss: 1.5347 - precision: 0.7632 -
recall: 0.2183 - val_accuracy: 0.2494 - val_auc: 0.7097 - val_loss: 2.5521 - val_precision: 0.4225 - val_recall:
0.1511
Epoch 4/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 778ms/step - accuracy: 0.6643 - auc: 0.9452 - loss: 0.8585 - precision: 0.8492 -
recall: 0.4863 - val_accuracy: 0.2469 - val_auc: 0.7254 - val_loss: 2.3152 - val_precision: 0.4337 - val_recall:
0.0907
Epoch 5/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 800ms/step - accuracy: 0.7698 - auc: 0.9671 - loss: 0.6320 - precision: 0.8951 -
recall: 0.6179 - val_accuracy: 0.2947 - val_auc: 0.7466 - val_loss: 2.6416 - val_precision: 0.4144 - val_recall:
0.1889
Epoch 6/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 780ms/step - accuracy: 0.8514 - auc: 0.9869 - loss: 0.3443 - precision: 0.9285 -
recall: 0.7753 - val_accuracy: 0.3199 - val_auc: 0.7573 - val_loss: 2.9091 - val_precision: 0.3852 - val_recall:
0.2368
Epoch 7/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 773ms/step - accuracy: 0.9152 - auc: 0.9955 - loss: 0.1932 - precision: 0.9607 -
recall: 0.8570 - val_accuracy: 0.3602 - val_auc: 0.7483 - val_loss: 3.9328 - val_precision: 0.4013 - val_recall:
0.3174
Epoch 8/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 778ms/step - accuracy: 0.9557 - auc: 0.9980 - loss: 0.1048 - precision: 0.9773 -
recall: 0.9271 - val_accuracy: 0.3552 - val_auc: 0.7391 - val_loss: 3.9011 - val_precision: 0.4082 - val_recall:
0.3249
Epoch 9/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 796ms/step - accuracy: 0.9765 - auc: 0.9993 - loss: 0.0628 - precision: 0.9866 -
recall: 0.9598 - val_accuracy: 0.3577 - val_auc: 0.7346 - val_loss: 3.8899 - val_precision: 0.3820 - val_recall:
0.3098
Epoch 10/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 776ms/step - accuracy: 0.9785 - auc: 0.9997 - loss: 0.0487 - precision: 0.9844 -
recall: 0.9658 - val_accuracy: 0.3401 - val_auc: 0.7331 - val_loss: 3.9741 - val_precision: 0.3795 - val_recall:
0.3174
Epoch 11/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 38s 757ms/step - accuracy: 0.9810 - auc: 0.9993 - loss: 0.0475 - precision: 0.9905 -
recall: 0.9680 - val_accuracy: 0.3602 - val_auc: 0.7371 - val_loss: 4.3465 - val_precision: 0.3964 - val_recall:
0.3325
Epoch 12/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 38s 769ms/step - accuracy: 0.9887 - auc: 0.9999 - loss: 0.0431 - precision: 0.9944 -
recall: 0.9825 - val_accuracy: 0.3627 - val_auc: 0.7314 - val_loss: 4.3610 - val_precision: 0.3897 - val_recall:
0.3249
Epoch 13/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 798ms/step - accuracy: 0.9886 - auc: 0.9998 - loss: 0.0460 - precision: 0.9905 -
recall: 0.9807 - val_accuracy: 0.3627 - val_auc: 0.7316 - val_loss: 4.6658 - val_precision: 0.3848 - val_recall:
0.3451
Epoch 14/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 792ms/step - accuracy: 0.9766 - auc: 0.9985 - loss: 0.0901 - precision: 0.9808 -
recall: 0.9684 - val_accuracy: 0.3678 - val_auc: 0.7316 - val_loss: 4.6920 - val_precision: 0.4000 - val_recall:
0.3375
Epoch 15/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 804ms/step - accuracy: 0.9869 - auc: 0.9993 - loss: 0.0532 - precision: 0.9921 -
recall: 0.9751 - val_accuracy: 0.3401 - val_auc: 0.7209 - val_loss: 4.5728 - val_precision: 0.3754 - val_recall:
0.3073
Training time: 590.58 seconds (9.84 minutes)
```

In [16]:
```
print("\nValidation Accuracy Model7")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history7.history[m][-1]:.4f}")
```

```
Validation Accuracy Model7
val_accuracy: 0.3401
val_precision: 0.3754
val_recall: 0.3073
val_auc: 0.7209
```

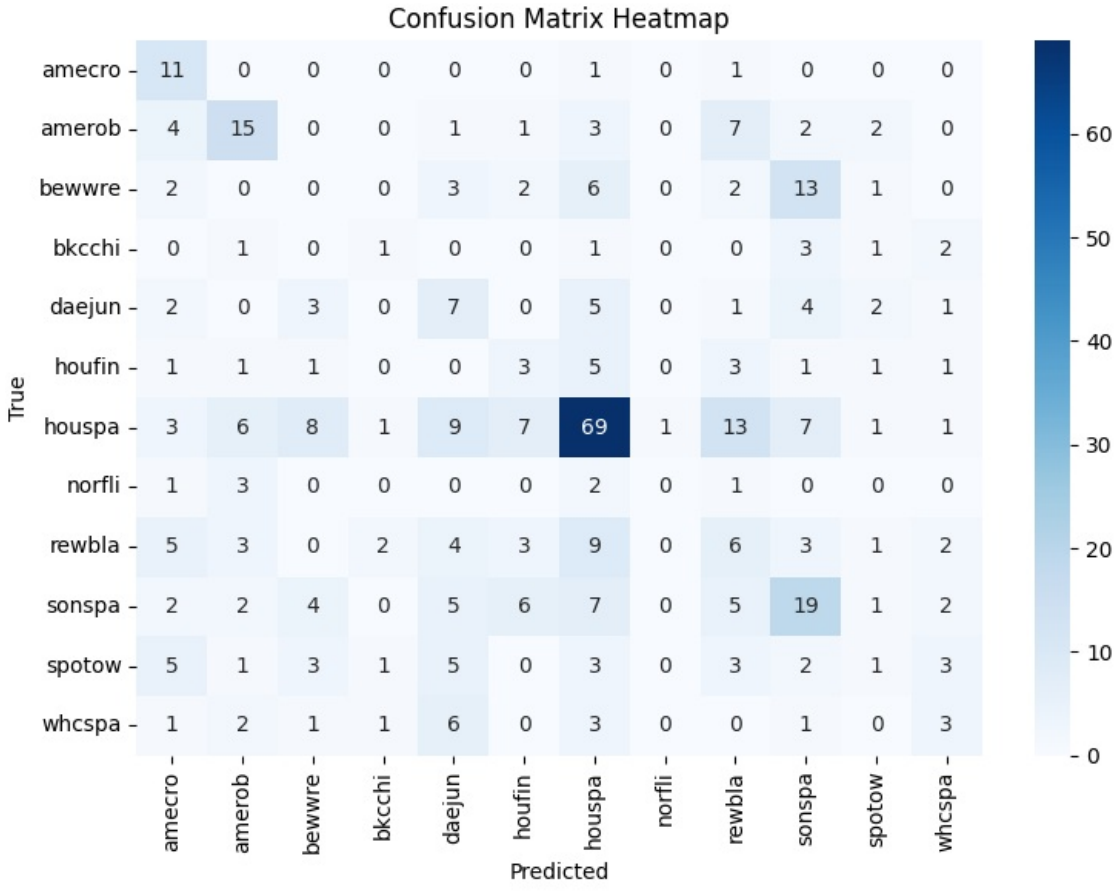
In [17]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history7.history['accuracy'], label='Train Accuracy')
plt.plot(history7.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history7.history['loss'], label='Train Loss')
```

```
plt.plot(history7.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



In [19]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
y_pred = model1b_weights.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm7 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm7, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 114ms/step
```



In [34]:
```python
# Defining model2
np.random.seed(42)
tf.random.set_seed(42)


model2_weights = Sequential()
model2_weights.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model2_weights.add(MaxPooling2D(pool_size=(2, 2)))

model2_weights.add(Conv2D(32, (3, 3), activation='relu'))
model2_weights.add(MaxPooling2D(pool_size=(2, 2)))

model2_weights.add(Flatten())
model2_weights.add(Dense(64, activation='relu'))
model2_weights.add(Dropout(0.3))
```

```
model2_weights.add(Dense(12, activation='softmax'))

model2_weights.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)
```

```
C:\Users\Nidhi\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [35]:
```
import time
start_time = time.time()
history9=model2_weights.fit(
    X_train, y_train,
    epochs=25,
    batch_size=128,
    validation_data=(X_test, y_test),
    class_weight=class_weights_dict
)

end_time = time.time()
time8 = end_time - start_time

print(f"Training time: {time8:.2f} seconds ({time8 / 60:.2f} minutes)")
```

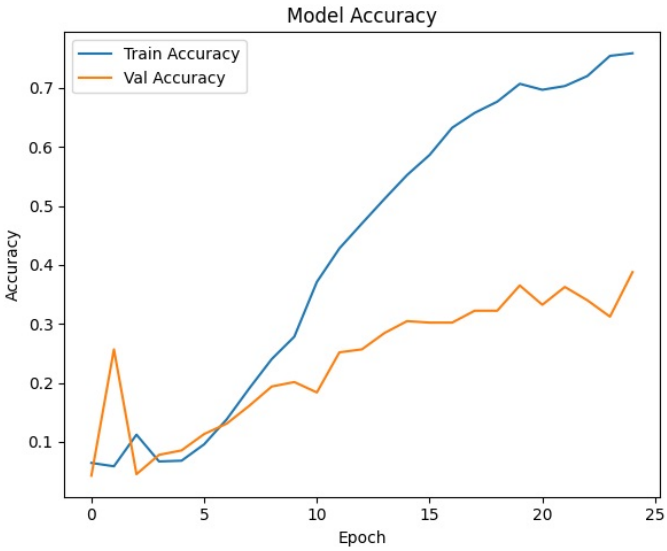
```
Epoch 1/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 11s 690ms/step - accuracy: 0.0689 - auc: 0.4961 - loss: 163.2697 - precision: 0.0691
- recall: 0.0642 - val_accuracy: 0.0428 - val_auc: 0.4999 - val_loss: 2.4839 - val_precision: 0.0000e+00 - val_r
ecall: 0.0000e+00
Epoch 2/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 672ms/step - accuracy: 0.0508 - auc: 0.5041 - loss: 2.5163 - precision: 0.0000e+00
- recall: 0.0000e+00 - val_accuracy: 0.2569 - val_auc: 0.5209 - val_loss: 2.4785 - val_precision: 0.0000e+00 - v
al_recall: 0.0000e+00
Epoch 3/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 640ms/step - accuracy: 0.1462 - auc: 0.5263 - loss: 2.5020 - precision: 0.0000e+00
- recall: 0.0000e+00 - val_accuracy: 0.0453 - val_auc: 0.5191 - val_loss: 2.4797 - val_precision: 0.0000e+00 - v
al_recall: 0.0000e+00
Epoch 4/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 653ms/step - accuracy: 0.0734 - auc: 0.5803 - loss: 2.4780 - precision: 0.0000e+00
- recall: 0.0000e+00 - val_accuracy: 0.0781 - val_auc: 0.6296 - val_loss: 2.4560 - val_precision: 0.0000e+00 - v
al_recall: 0.0000e+00
Epoch 5/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 652ms/step - accuracy: 0.0638 - auc: 0.6437 - loss: 2.4515 - precision: 0.0000e+00
- recall: 0.0000e+00 - val_accuracy: 0.0856 - val_auc: 0.6300 - val_loss: 2.4368 - val_precision: 0.0000e+00 - v
al_recall: 0.0000e+00
Epoch 6/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 648ms/step - accuracy: 0.0890 - auc: 0.6607 - loss: 2.3678 - precision: 0.0000e+00
- recall: 0.0000e+00 - val_accuracy: 0.1134 - val_auc: 0.6442 - val_loss: 2.4103 - val_precision: 0.0000e+00 - v
al_recall: 0.0000e+00
Epoch 7/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 638ms/step - accuracy: 0.1319 - auc: 0.7008 - loss: 2.2343 - precision: 0.5258 - r
ecall: 0.0028 - val_accuracy: 0.1310 - val_auc: 0.6520 - val_loss: 2.3727 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 8/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 652ms/step - accuracy: 0.1816 - auc: 0.7435 - loss: 2.0882 - precision: 0.7431 - r
ecall: 0.0162 - val_accuracy: 0.1612 - val_auc: 0.6643 - val_loss: 2.3395 - val_precision: 0.0000e+00 - val_reca
ll: 0.0000e+00
Epoch 9/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 630ms/step - accuracy: 0.2214 - auc: 0.7636 - loss: 1.9386 - precision: 0.7876 - r
ecall: 0.0336 - val_accuracy: 0.1940 - val_auc: 0.6988 - val_loss: 2.2705 - val_precision: 0.4444 - val_recall:
0.0101
Epoch 10/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 654ms/step - accuracy: 0.2593 - auc: 0.7997 - loss: 1.7938 - precision: 0.7355 - r
ecall: 0.0707 - val_accuracy: 0.2015 - val_auc: 0.7127 - val_loss: 2.3353 - val_precision: 0.4857 - val_recall:
0.0428
Epoch 11/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 634ms/step - accuracy: 0.3495 - auc: 0.8241 - loss: 1.6483 - precision: 0.7379 - r
ecall: 0.1487 - val_accuracy: 0.1839 - val_auc: 0.6809 - val_loss: 2.4259 - val_precision: 0.3333 - val_recall:
0.0403
Epoch 12/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 633ms/step - accuracy: 0.3874 - auc: 0.8484 - loss: 1.4748 - precision: 0.7241 - r
ecall: 0.1949 - val_accuracy: 0.2519 - val_auc: 0.7337 - val_loss: 2.2310 - val_precision: 0.4000 - val_recall:
0.0403
Epoch 13/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 636ms/step - accuracy: 0.4616 - auc: 0.8888 - loss: 1.2352 - precision: 0.8305 - r
ecall: 0.2657 - val_accuracy: 0.2569 - val_auc: 0.7369 - val_loss: 2.2863 - val_precision: 0.4478 - val_recall:
```

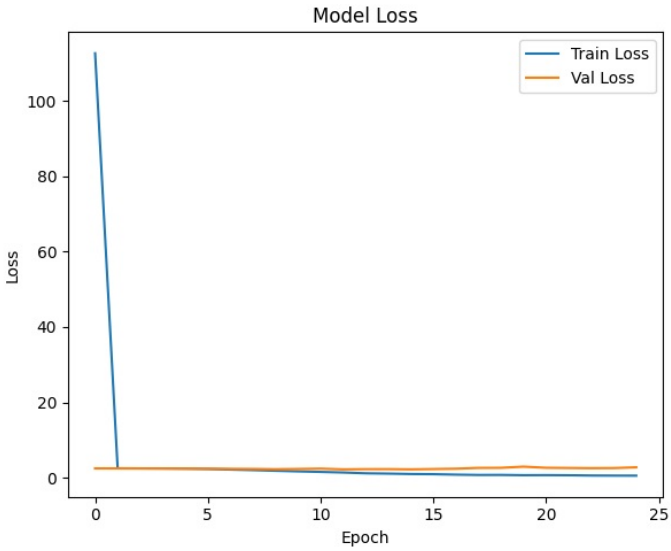
```
0.0756
Epoch 14/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 639ms/step - accuracy: 0.5025 - auc: 0.9061 - loss: 1.1478 - precision: 0.8069 - r
ecall: 0.3071 - val_accuracy: 0.2846 - val_auc: 0.7430 - val_loss: 2.2941 - val_precision: 0.4255 - val_recall:
0.1008
Epoch 15/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 630ms/step - accuracy: 0.5384 - auc: 0.9256 - loss: 1.0337 - precision: 0.8381 - r
ecall: 0.3461 - val_accuracy: 0.3048 - val_auc: 0.7619 - val_loss: 2.2328 - val_precision: 0.4949 - val_recall:
0.1234
Epoch 16/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 645ms/step - accuracy: 0.5882 - auc: 0.9414 - loss: 0.9326 - precision: 0.8378 - r
ecall: 0.4139 - val_accuracy: 0.3023 - val_auc: 0.7431 - val_loss: 2.3148 - val_precision: 0.5366 - val_recall:
0.1108
Epoch 17/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 647ms/step - accuracy: 0.6093 - auc: 0.9452 - loss: 0.8621 - precision: 0.9035 - r
ecall: 0.4341 - val_accuracy: 0.3023 - val_auc: 0.7634 - val_loss: 2.4053 - val_precision: 0.4236 - val_recall:
0.1537
Epoch 18/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 631ms/step - accuracy: 0.6514 - auc: 0.9581 - loss: 0.7557 - precision: 0.8517 - r
ecall: 0.5075 - val_accuracy: 0.3224 - val_auc: 0.7524 - val_loss: 2.6215 - val_precision: 0.3795 - val_recall:
0.1587
Epoch 19/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 646ms/step - accuracy: 0.6622 - auc: 0.9629 - loss: 0.7551 - precision: 0.8698 - r
ecall: 0.5303 - val_accuracy: 0.3224 - val_auc: 0.7515 - val_loss: 2.6418 - val_precision: 0.4000 - val_recall:
0.1713
Epoch 20/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 630ms/step - accuracy: 0.6908 - auc: 0.9673 - loss: 0.7316 - precision: 0.8741 - r
ecall: 0.5149 - val_accuracy: 0.3652 - val_auc: 0.7539 - val_loss: 2.9539 - val_precision: 0.4508 - val_recall:
0.2771
Epoch 21/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 653ms/step - accuracy: 0.7017 - auc: 0.9692 - loss: 0.7035 - precision: 0.8686 - r
ecall: 0.5755 - val_accuracy: 0.3325 - val_auc: 0.7609 - val_loss: 2.6510 - val_precision: 0.4375 - val_recall:
0.2116
Epoch 22/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 625ms/step - accuracy: 0.7068 - auc: 0.9697 - loss: 0.6671 - precision: 0.8609 - r
ecall: 0.5762 - val_accuracy: 0.3627 - val_auc: 0.7546 - val_loss: 2.5958 - val_precision: 0.4749 - val_recall:
0.2141
Epoch 23/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 643ms/step - accuracy: 0.7264 - auc: 0.9765 - loss: 0.5769 - precision: 0.8618 - r
ecall: 0.6319 - val_accuracy: 0.3401 - val_auc: 0.7631 - val_loss: 2.5409 - val_precision: 0.4551 - val_recall:
0.2040
Epoch 24/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 8s 640ms/step - accuracy: 0.7524 - auc: 0.9780 - loss: 0.5960 - precision: 0.8717 - r
ecall: 0.6470 - val_accuracy: 0.3123 - val_auc: 0.7560 - val_loss: 2.5722 - val_precision: 0.4362 - val_recall:
0.1637
Epoch 25/25
13/13 ━━━━━━━━━━━━━━━━━━━━ 9s 648ms/step - accuracy: 0.7487 - auc: 0.9775 - loss: 0.5744 - precision: 0.8992 - r
ecall: 0.6270 - val_accuracy: 0.3879 - val_auc: 0.7687 - val_loss: 2.7792 - val_precision: 0.4783 - val_recall:
0.2771
Training time: 213.46 seconds (3.56 minutes)
```

In [38]:
```python
print("\nValidation Accuracy Model8")
for m in ['val_accuracy', 'val_precision', 'val_recall', 'val_auc']:
    print(f"{m}: {history9.history[m][-1]:.4f}")
```

```
Validation Accuracy Model8
val_accuracy: 0.3879
val_precision: 0.4783
val_recall: 0.2771
val_auc: 0.7687
```

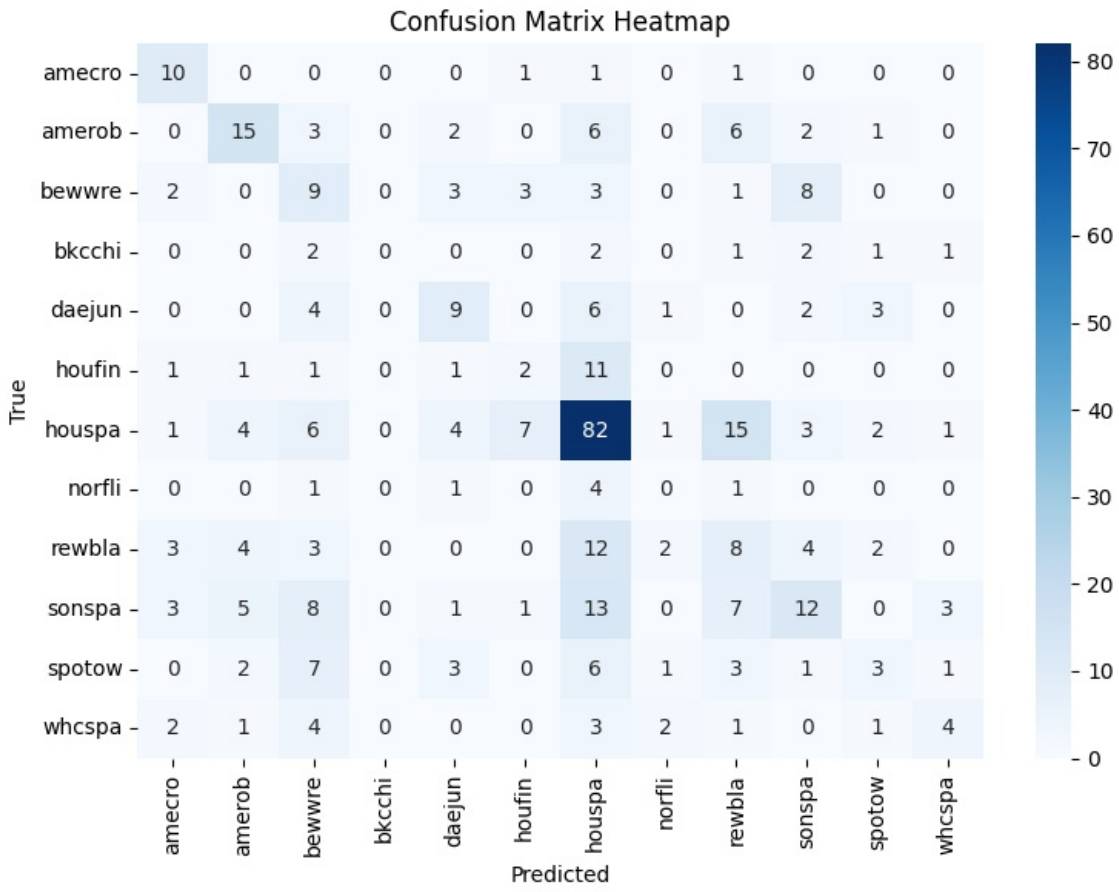
In [36]:
```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history9.history['accuracy'], label='Train Accuracy')
plt.plot(history9.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history9.history['loss'], label='Train Loss')
plt.plot(history9.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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





In [37]:
```
y_pred = model2_weights.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm8 = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm8, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step
```



In [46]:
```
# saving bestmodel out of all
model1b_multi.save("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\\Homework3\\bestmod
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [47]:
```
model2_multi.save("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\Homework3\\bestmode
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [39]:
```
model2_weights.save("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\Homework3\\bestmo
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

External Test Data

In [48]:
```
# Loading required libraries
import librosa
import librosa.display
import os
```

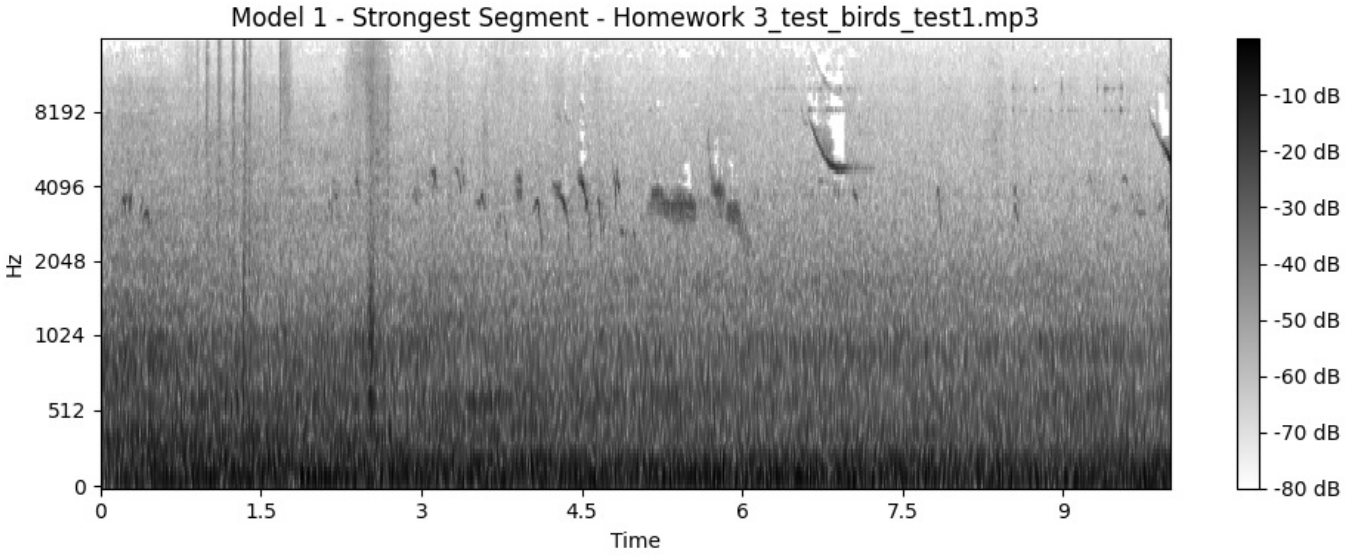
In [41]:
```
import librosa
import numpy as np
import tensorflow as tf
import os
from collections import Counter


SAMPLE_RATE    = 32000
WINDOW_SIZE    = 10
STEP_SIZE      = 1.5
N_MELS         = 128
HOP_LENGTH     = 128
WIN_LENGTH     = 512
EXPECTED_WIDTH = 517

# ——— Labels & Models ———————————————————————————————
species_labels = species  # your list of 12 species names

model1 = tf.keras.models.load_model("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\H
model2 = tf.keras.models.load_model("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\H
model3 = tf.keras.models.load_model("C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\H
models = {"Model 1": model1, "Model 2": model2,"Model 3": model3}

mp3_files = [
    "C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\Homework3\\Homework 3_test_birds_
    "C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\Homework3\\Homework 3_test_birds_
    "C:\\Users\\Nidhi\\Documents\\College Material\\Spring Quarter 2025\\SML2\Homework3\\Homework 3_test_birds_
]

def sliding_window_preds(audio, sr, model):
    window_len = int(sr * WINDOW_SIZE)
    step_len   = int(sr * STEP_SIZE)
    num_windows = max(1, int((len(audio) - window_len) / step_len) + 1)
    all_preds = []

    for i in range(num_windows):
        start, end = i*step_len, i*step_len + window_len
        y_win = audio[start:end].copy()

        # spectrogram
        mel = librosa.feature.melspectrogram(
            y=y_win, sr=sr,
            n_mels=N_MELS,
            hop_length=HOP_LENGTH,
            win_length=WIN_LENGTH
        )
        mel_db = librosa.power_to_db(mel, ref=np.max)
        norm  = (mel_db - mel_db.min()) / (mel_db.max() - mel_db.min())

        # resize to EXPECTED_WIDTH
        if norm.shape[1] > EXPECTED_WIDTH:
            norm = norm[:, :EXPECTED_WIDTH]
        else:
            pad_width = EXPECTED_WIDTH - norm.shape[1]
            norm = np.pad(norm, ((0,0),(0,pad_width)), mode='constant')

        # predict
        inp   = np.expand_dims(norm, axis=(0,-1))  # shape: (1,128,517,1)
        pred  = model.predict(inp, verbose=0)[0]   # vector of 12 probs
        all_preds.append(pred)

    return np.stack(all_preds)  # shape: (num_windows, 12)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` w
ill be empty until you train or evaluate the model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` w
ill be empty until you train or evaluate the model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` w
ill be empty until you train or evaluate the model.
```

In [43]:

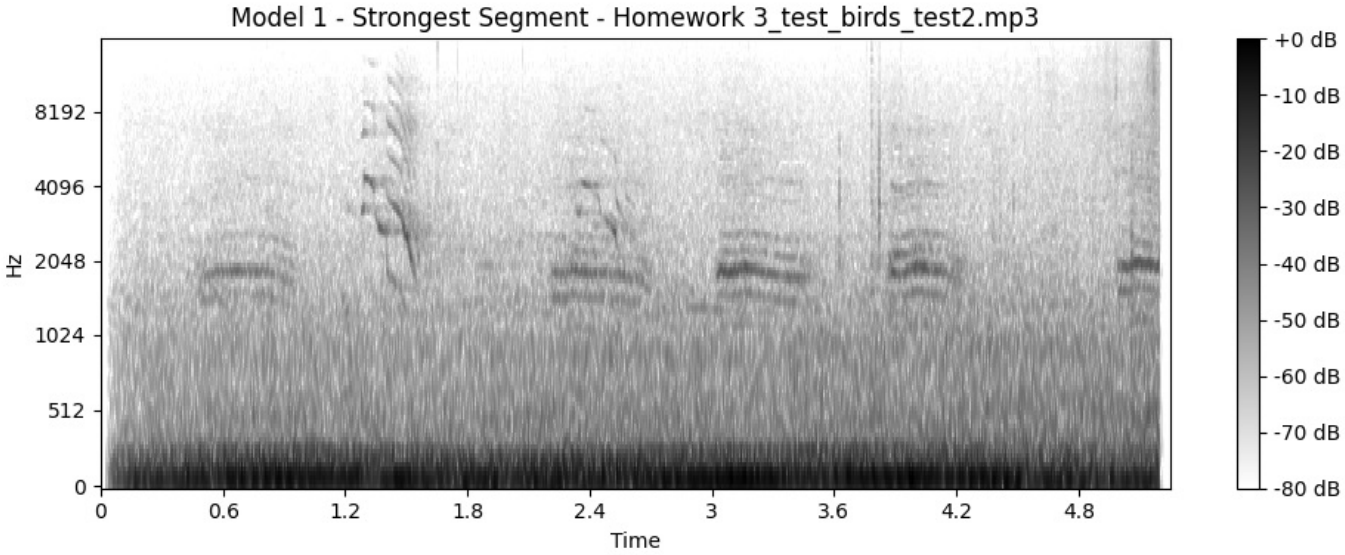
```python
for model_idx, model in enumerate([model1, model2,model3], start=1):
    print(f"\n============================")
    print(f"Predictions using Model {model_idx}")
    print(f"============================")

    for file in mp3_files:
        print(f"\nFile: {os.path.basename(file)}")

        y, sr = librosa.load(file, sr=SAMPLE_RATE)
        preds = sliding_window_preds(y, sr, model)


        # Average and top predictions
        avg_pred = np.mean(preds, axis=0)
        top3_avg = np.argsort(avg_pred)[::-1][:3]

        print("Top 3 Average Predictions:")
        for i in top3_avg:
            print(f" - {species_labels[i]}: {avg_pred[i]:.2f}")

        # Count most frequent top-1 prediction per window
        top1_per_window = np.argmax(preds, axis=1)
        counts = Counter(top1_per_window)
        print("\nTop species per window:")
        for idx, cnt in counts.most_common():
            print(f" - {species_labels[idx]}: {cnt} windows")

        # Plot the segment with the strongest prediction
        best_idx = np.argmax(np.max(preds, axis=1))
        best_start = int(best_idx * SAMPLE_RATE * STEP_SIZE)
        best_end = best_start + int(SAMPLE_RATE * WINDOW_SIZE)
        y_best = y[best_start:best_end]

        mel = librosa.feature.melspectrogram(y=y_best, sr=sr, n_mels=N_MELS,
                                             hop_length=HOP_LENGTH, win_length=WIN_LENGTH)
        mel_db = librosa.power_to_db(mel, ref=np.max)

        plt.figure(figsize=(10, 4))
        librosa.display.specshow(mel_db, sr=sr, x_axis='time', y_axis='mel',
                                 cmap='gray_r', hop_length=HOP_LENGTH)
        plt.colorbar(format='%+2.0f dB')
        plt.title(f"Model {model_idx} - Strongest Segment - {os.path.basename(file)}")
        plt.tight_layout()
        plt.show()
```

```
============================
Predictions using Model 1
============================

File: Homework 3_test_birds_test1.mp3
Top 3 Average Predictions:
 - rewbla: 0.84
 - sonspa: 0.07
 - daejun: 0.06

Top species per window:
 - rewbla: 9 windows
```



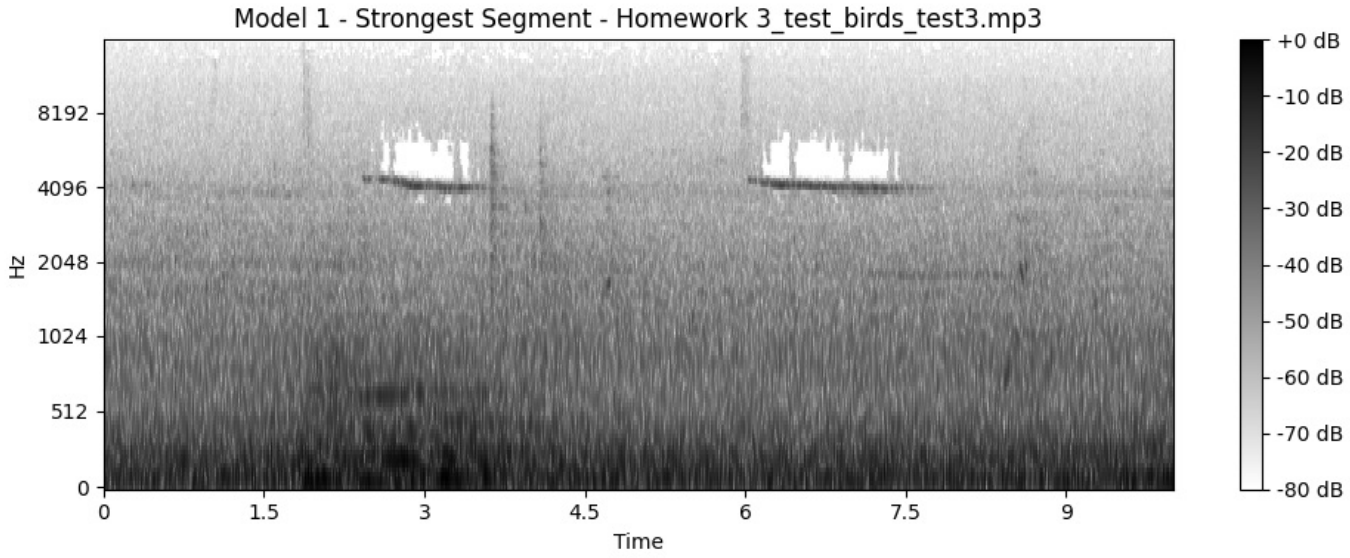
File: Homework 3_test_birds_test2.mp3
Top 3 Average Predictions:
 - rewbla: 0.59
 - daejun: 0.16
 - sonspa: 0.14

Top species per window:
 - rewbla: 1 windows

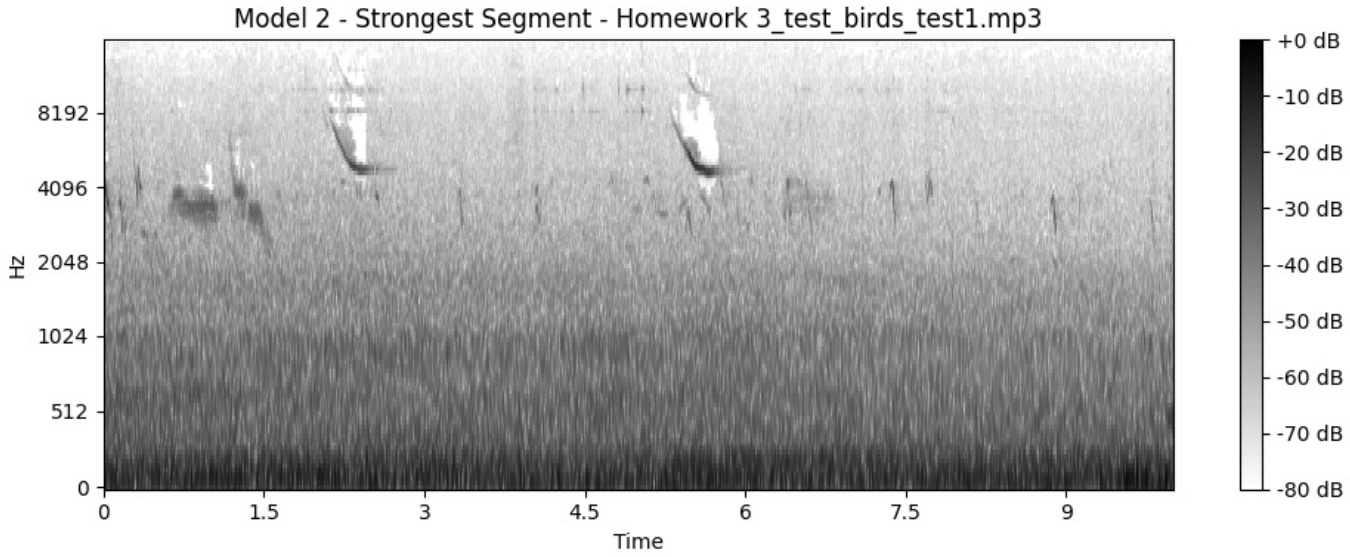


File: Homework 3_test_birds_test3.mp3
Top 3 Average Predictions:
 - rewbla: 0.92
 - daejun: 0.04
 - sonspa: 0.03

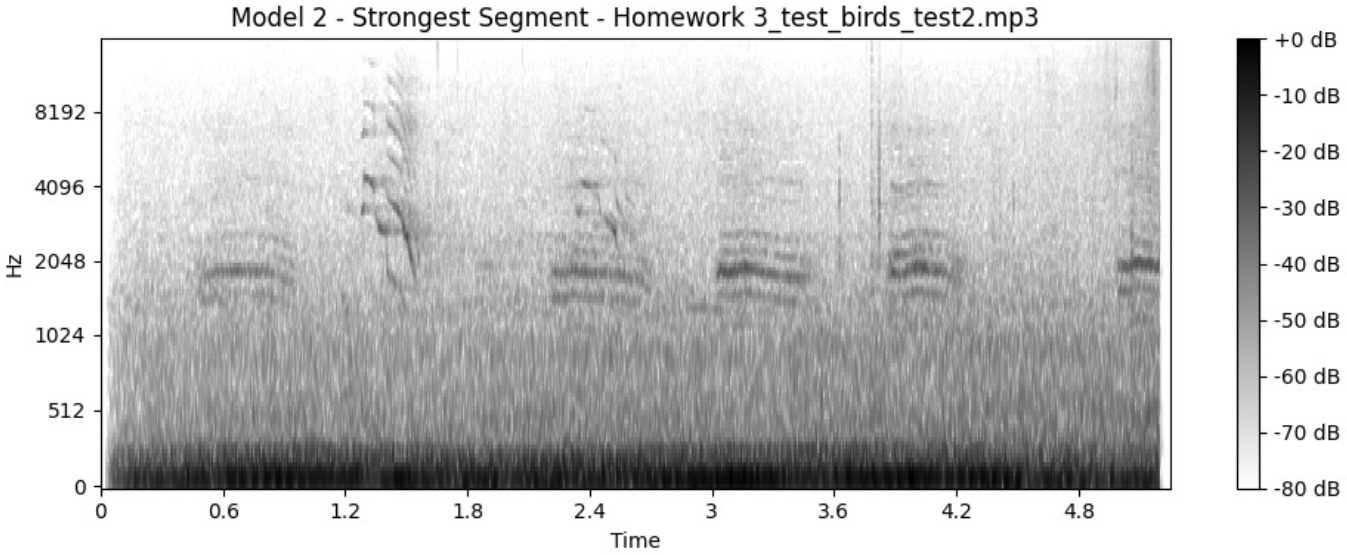
Top species per window:
 - rewbla: 4 windows




```
============================
Predictions using Model 2
============================

File: Homework 3_test_birds_test1.mp3
Top 3 Average Predictions:
 - houspa: 0.45
 - sonspa: 0.28
 - houfin: 0.27

Top species per window:
 - houspa: 9 windows
```

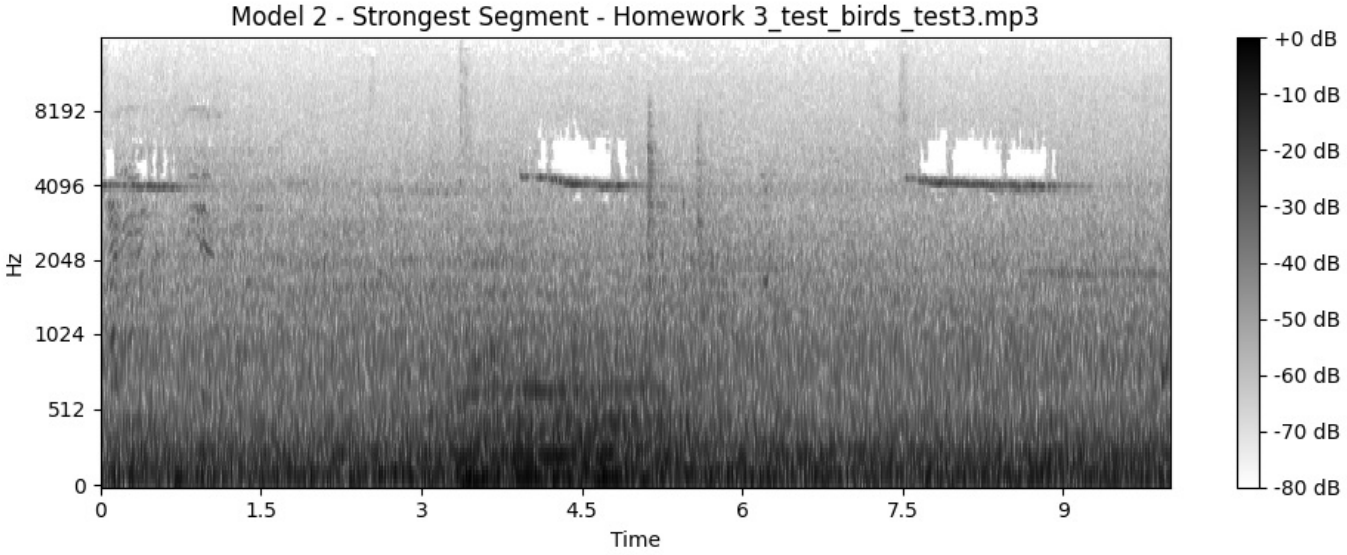



```
File: Homework 3_test_birds_test2.mp3
Top 3 Average Predictions:
 - houspa: 0.42
 - sonspa: 0.29
 - houfin: 0.28

Top species per window:
 - houspa: 1 windows
```



File: Homework 3_test_birds_test3.mp3
Top 3 Average Predictions:
 - houspa: 0.48
 - sonspa: 0.27
 - houfin: 0.25

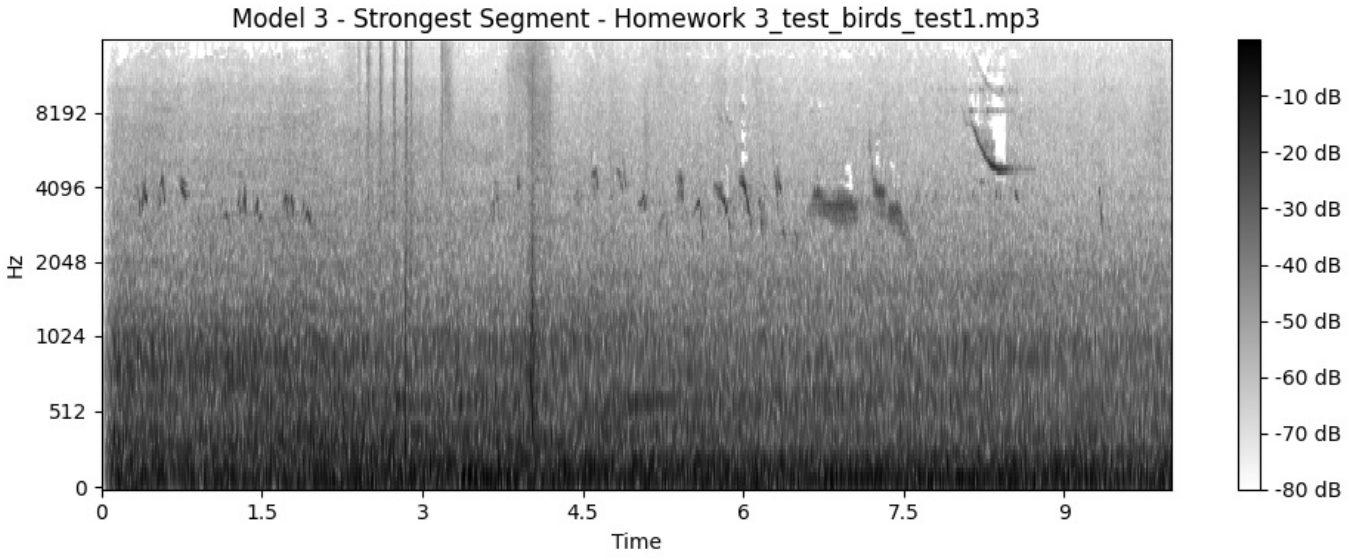
Top species per window:
 - houspa: 4 windows



==========
Predictions using Model 3
==========

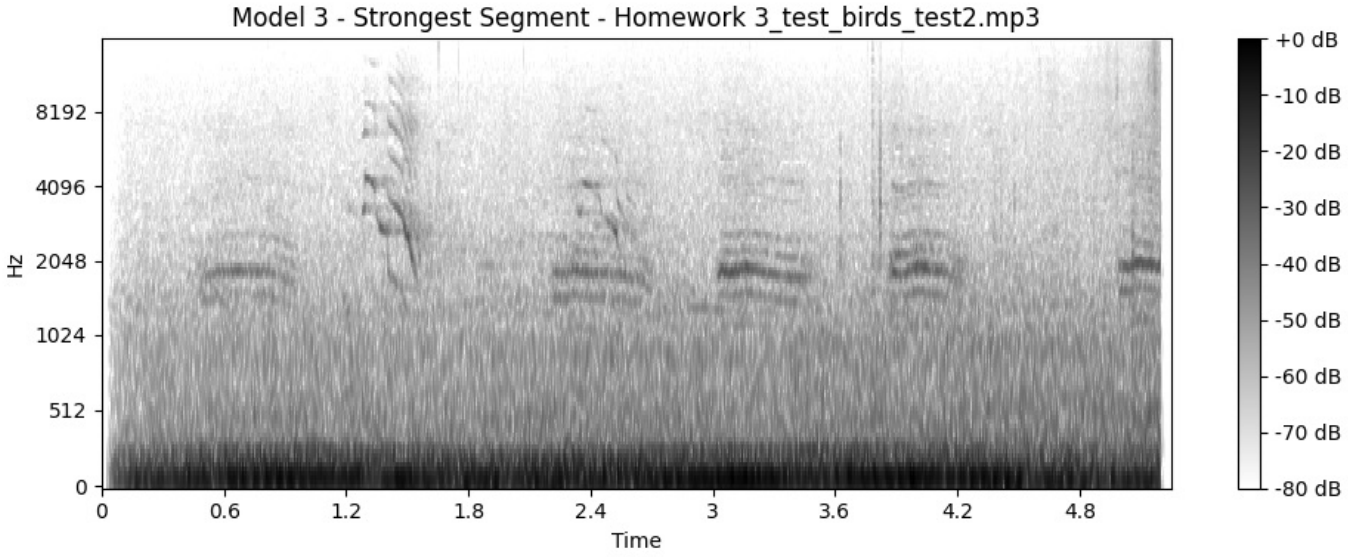
File: Homework 3_test_birds_test1.mp3
Top 3 Average Predictions:
 - rewbla: 0.27
 - amerob: 0.21
 - amecro: 0.12

Top species per window:
 - rewbla: 9 windows



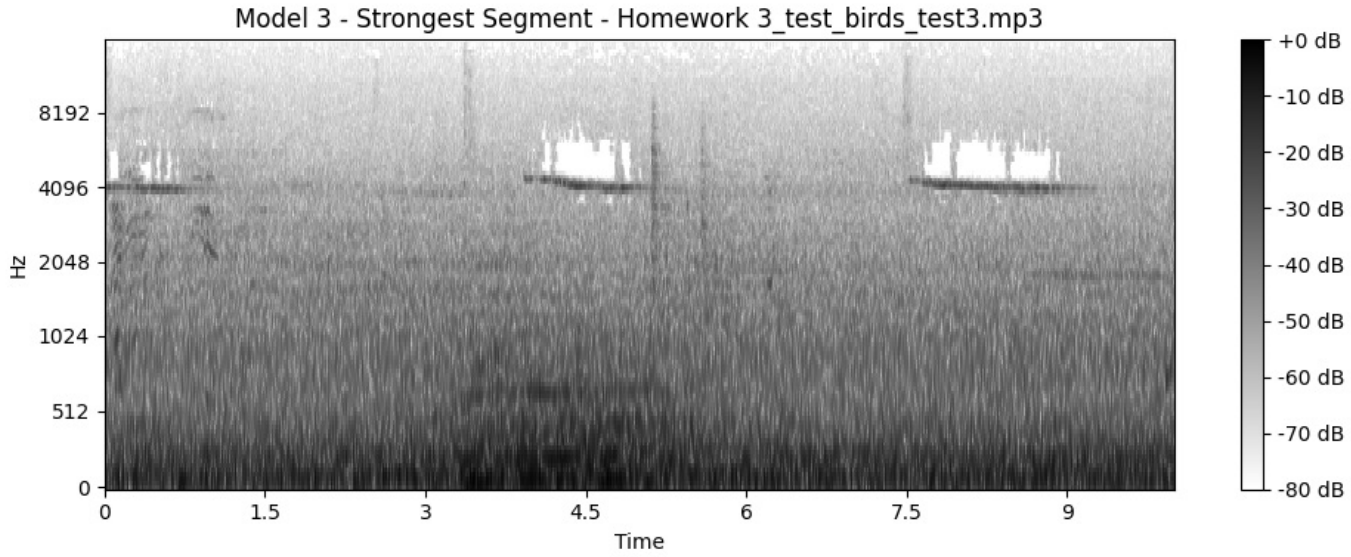
File: Homework 3_test_birds_test2.mp3
Top 3 Average Predictions:
 - rewbla: 0.21
 - amerob: 0.21
 - amecro: 0.14

Top species per window:
 - rewbla: 1 windows



File: Homework 3_test_birds_test3.mp3
Top 3 Average Predictions:
 - rewbla: 0.26
 - amerob: 0.23
 - amecro: 0.12

Top species per window:
 - amerob: 2 windows
 - rewbla: 2 windows


Model 3 - Strongest Segment - Homework 3_test_birds_test3.mp3
Hz
8192
4096
2048
1024
512
0
0
1.5
3
4.5
6
7.5
9
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

In [ ]: