

```
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 from 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)
```

	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

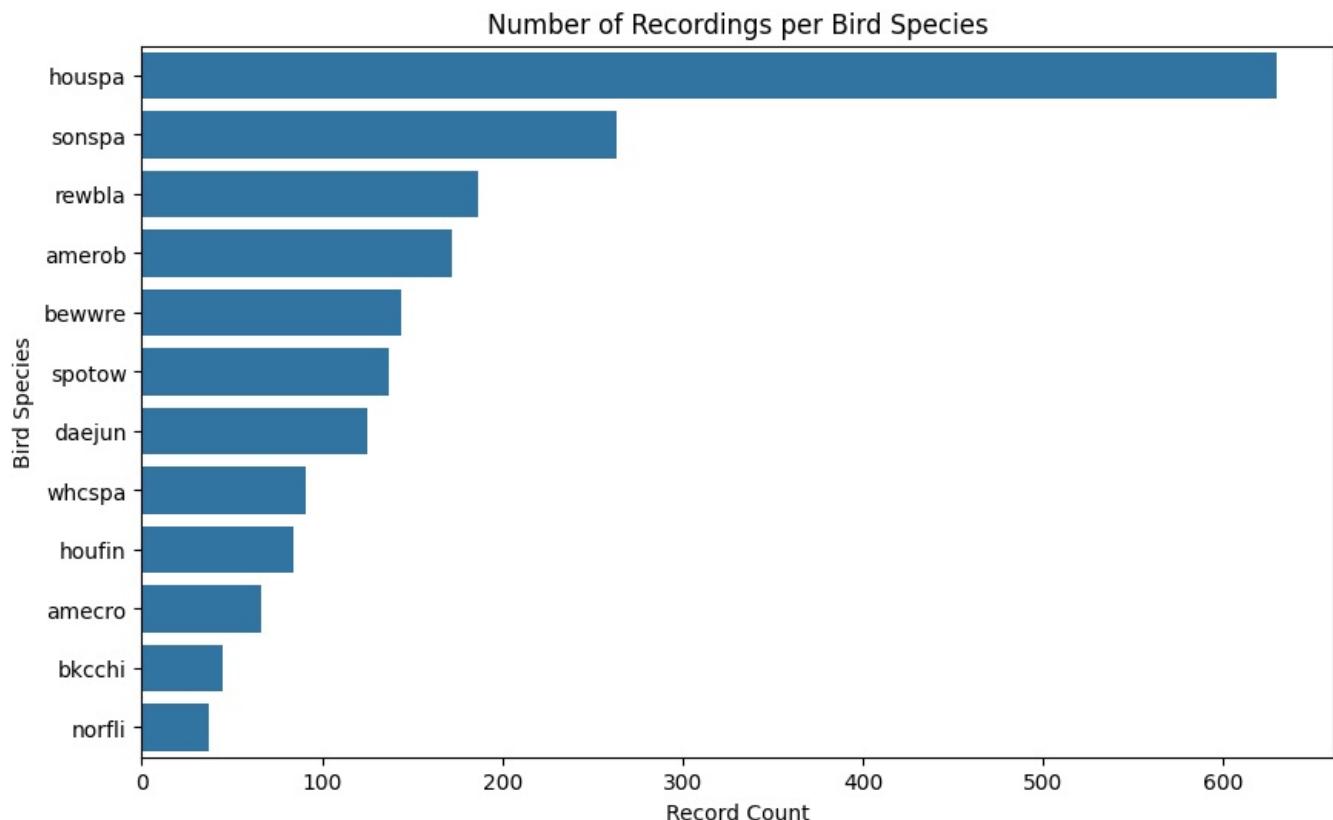
```
In [4]: df.describe()
```

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()
```



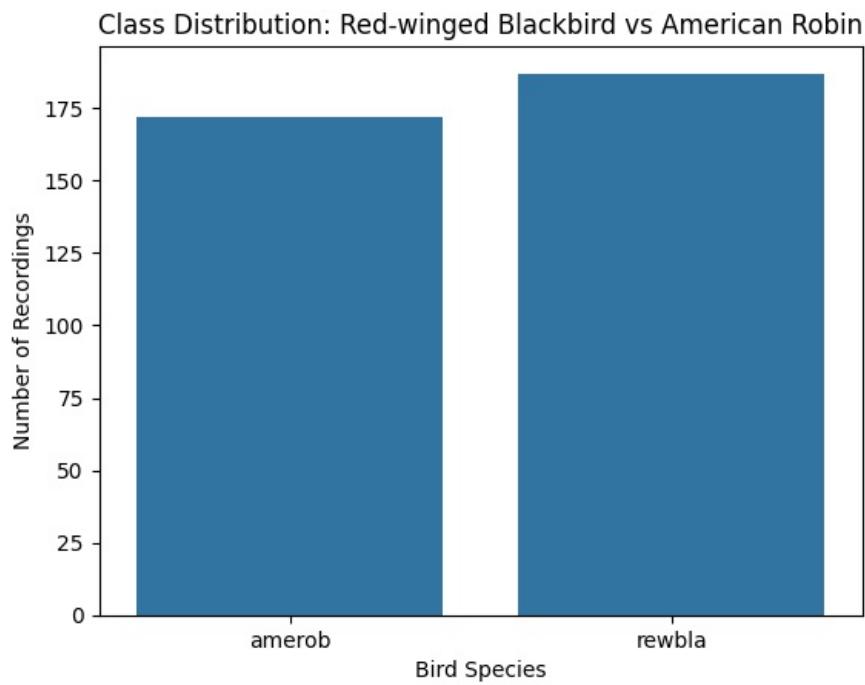
- From the plot, majority samples are of house sparrow. While, Northern flicker and Black-capped chickadee is in minority class.
- We will perform binary classification on Red-winged blackbird and American Robin as they have around same samples.

In [8]:

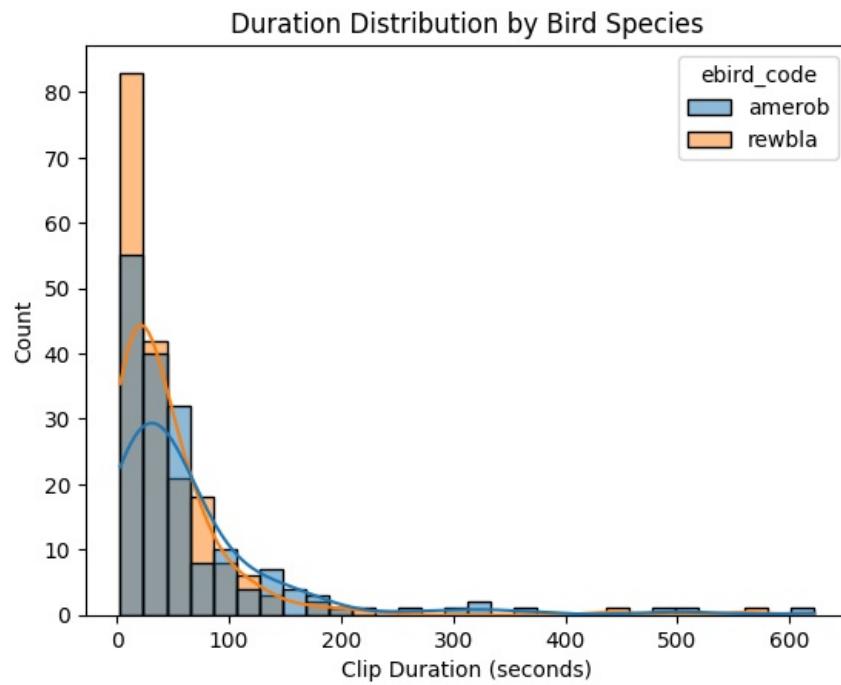
```
binary_df = df[df['ebird_code'].isin(['rewbla', 'amerob'])].copy()
```

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. Splitting 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: Preparing 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]: # Concatenating 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: 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)
```

## 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 - recall: 0.6092 - val\_accuracy: 0.4815 - val\_auc: 0.5000 - val\_loss: 252.3771 - val\_precision: 0.0000e+00 - val\_recall: 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\_recall: 1.0000  
Epoch 3/50  
4/4 3s 741ms/step - accuracy: 0.4760 - auc: 0.4165 - loss: 35.9204 - precision: 0.4752 - recall: 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 - recall: 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 - recall: 0.8459 - val\_accuracy: 0.4815 - val\_auc: 0.5000 - val\_loss: 10.3164 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00  
Epoch 6/50  
4/4 4s 940ms/step - accuracy: 0.5532 - auc: 0.5359 - loss: 8.9385 - precision: 0.5595 - recall: 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 - recall: 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 - recall: 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 - recall: 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: 1.0000  
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: 1.0000  
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: 0.1071  
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: 0.8393  
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: 0.6964

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: 0.8393

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: 0.6071

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: 0.6429

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: 0.6607

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: 0.6250

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: 0.7857

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: 0.7500

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: 0.6071

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: 0.5000

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: 0.5536

Epoch 25/50  
4/4 4s 987ms/step - accuracy: 0.9890 - auc: 0.9997 - loss: 0.0331 - precision: 0.9918 - recall: 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: 0.6786

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: 0.6429

Epoch 28/50  
4/4 4s 980ms/step - accuracy: 0.9843 - auc: 0.9999 - loss: 0.0245 - precision: 0.9765 - recall: 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 - recall: 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: 0.8036

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: 0.8393

Epoch 32/50  
4/4 4s 987ms/step - accuracy: 0.9916 - auc: 0.9996 - loss: 0.0223 - precision: 0.9917 - recall: 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: 0.7857

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: 0.7679

Epoch 35/50  
4/4 4s 936ms/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0176 - precision: 1.0000 - recall: 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 - recall: 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 - recall: 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: 0.7143  
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: 0.6964  
Epoch 40/50  
4/4 3s 939ms/step - accuracy: 0.9958 - auc: 0.9999 - loss: 0.0166 - precision: 0.9920 - recall: 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: 0.6429  
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: 0.5000  
Epoch 43/50  
4/4 4s 995ms/step - accuracy: 0.9958 - auc: 1.0000 - loss: 0.0125 - precision: 1.0000 - recall: 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: 0.7500  
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: 0.7679  
Epoch 46/50  
4/4 4s 958ms/step - accuracy: 0.9858 - auc: 0.9997 - loss: 0.0279 - precision: 0.9874 - recall: 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 - recall: 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: 0.7679  
Epoch 49/50  
4/4 4s 977ms/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0047 - precision: 1.0000 - recall: 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 - recall: 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)
```

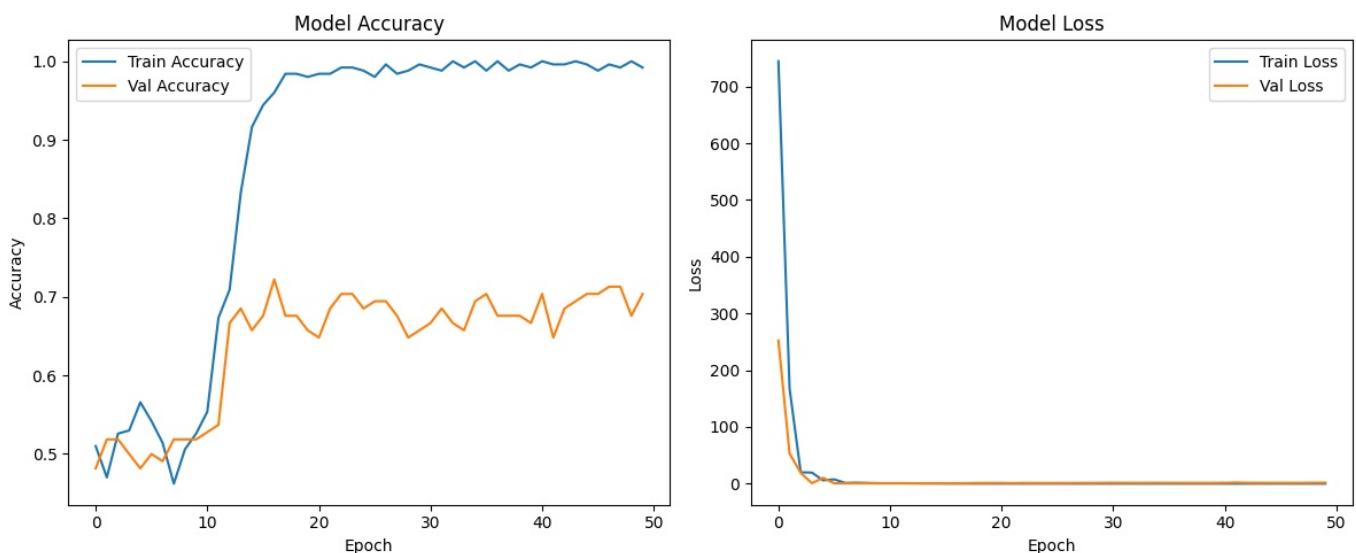
In [21]:

```
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

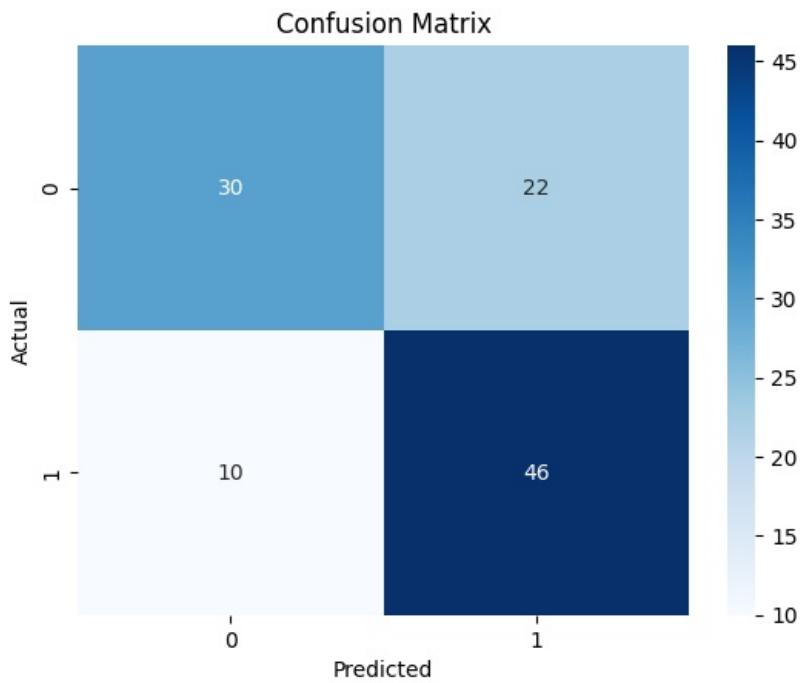
loss1, accuracy1, precision1, recall1, auc1 = model1_binary.evaluate(X_test, y_test)

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 - recall: 0.8099
Test Accuracy: 0.7037
Test Precision: 0.6765
Test Recall: 0.8214
Test AUC: 0.7618
4/4 ━━━━━━━━ 1s 210ms/step
```

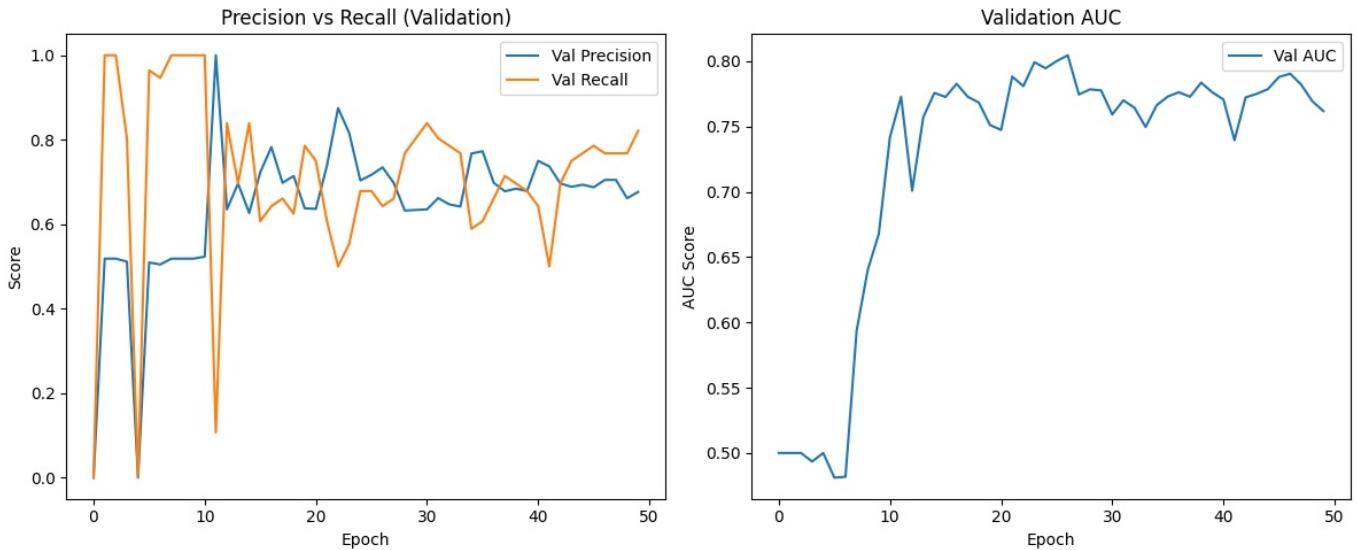


- The heatmap shows that the model correctly predicted most Red-winged Blackbird calls (46).
- However, some American Robin samples were misclassified as Red-winged Blackbird (22).

```
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, indicating that the model is good in prediction.

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: 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 [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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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))

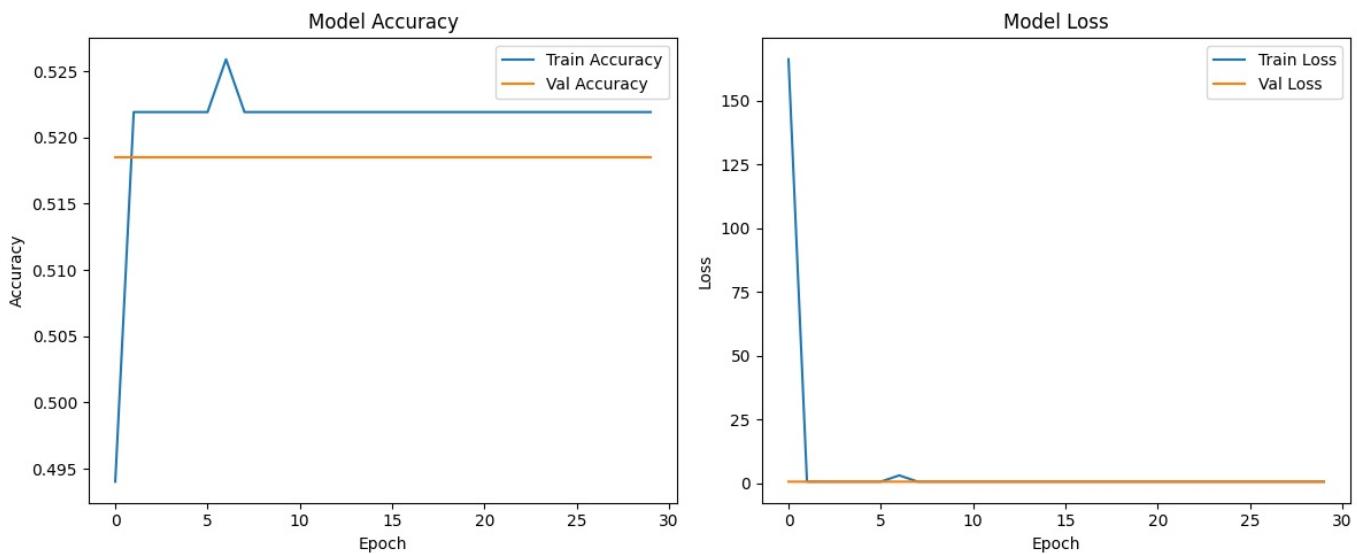
```

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)

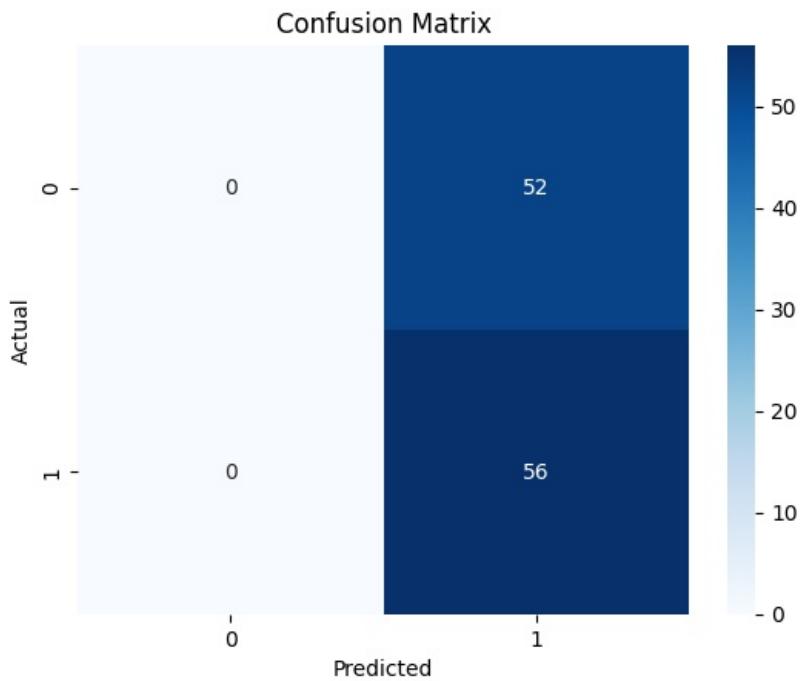
print(f"Test Accuracy: {accuracy1:.4f}")
print(f"Test Precision: {precision1:.4f}")
print(f"Test Recall: {recall1:.4f}")
print(f"Test AUC: {auc1:.4f}")

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 - recall: 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 - recall: 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 - recall: 0.9095 - val_accuracy: 0.4907 - val_auc: 0.5019 - val_loss: 0.8928 - val_precision: 0.6000 - val_recall: 0.0536
Epoch 3/30
4/4 11s 2s/step - accuracy: 0.4385 - auc: 0.4686 - loss: 8.0707 - precision: 0.4917 - recall: 0.3801 - val_accuracy: 0.6389 - val_auc: 0.7160 - val_loss: 0.6668 - val_precision: 0.6104 - val_recall: 0.8393
Epoch 4/30
4/4 10s 3s/step - accuracy: 0.6946 - auc: 0.6854 - loss: 0.8393 - precision: 0.6713 - recall: 0.8223 - val_accuracy: 0.5648 - val_auc: 0.6690 - val_loss: 0.7297 - val_precision: 0.7368 - val_recall: 0.2500
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: 0.1250
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: 0.7679
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: 0.9107
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: 0.7857
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: 0.0179
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: 0.3036
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: 0.6607
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: 0.6429
Epoch 14/30
4/4 10s 2s/step - accuracy: 0.9968 - auc: 1.0000 - loss: 0.0434 - precision: 0.9940 - recall: 1.0000 - val_accuracy: 0.6296 - val_auc: 0.7148 - val_loss: 1.6698 - val_precision: 0.7857 - val_recall: 0.3929
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: 0.4464
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: 0.5893
Epoch 17/30
4/4 11s 3s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0130 - precision: 1.0000 - recall: 1.0000 - val_accuracy: 0.6667 - val_auc: 0.7399 - val_loss: 1.1568 - val_precision: 0.6923 - val_recall: 0.6429
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: 0.5536
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: 0.4643
Epoch 20/30
4/4 9s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0040 - precision: 1.0000 - recall

```

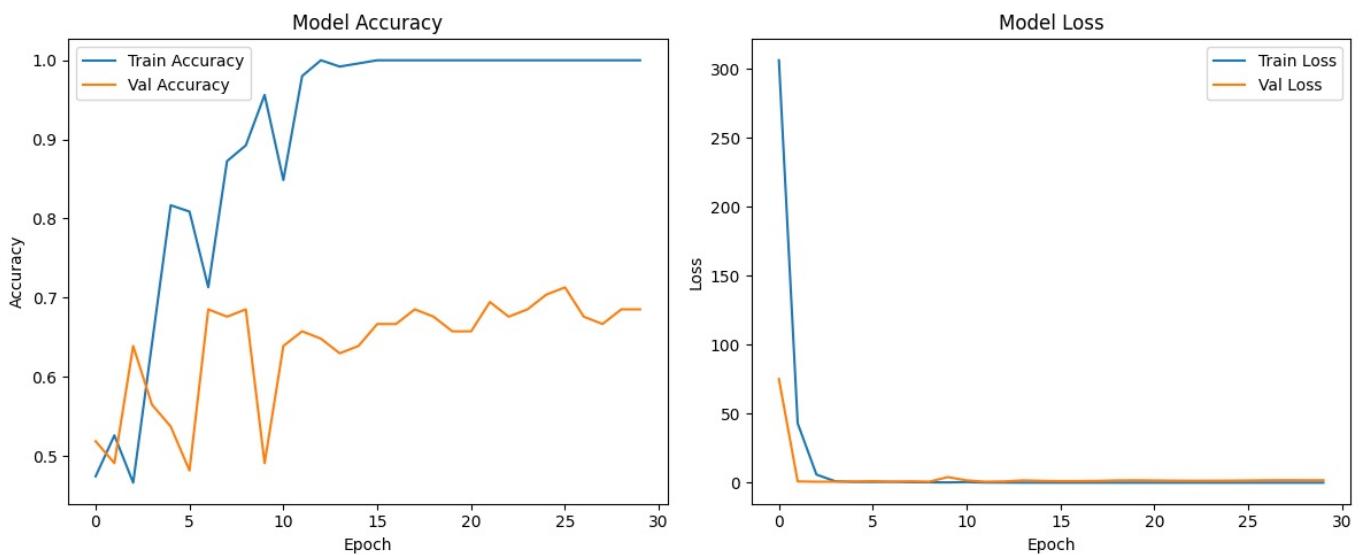
```
: 1.0000 - val_accuracy: 0.6574 - val_auc: 0.7643 - val_loss: 1.7229 - val_precision: 0.8065 - val_recall: 0.446
4
Epoch 21/30
4/4 8s 2s/step - accuracy: 1.0000 - auc: 1.0000 - loss: 0.0022 - precision: 1.0000 - recall: 1.0000 - val_accuracy: 0.6574 - val_auc: 0.7577 - val_loss: 1.5974 - val_precision: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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.

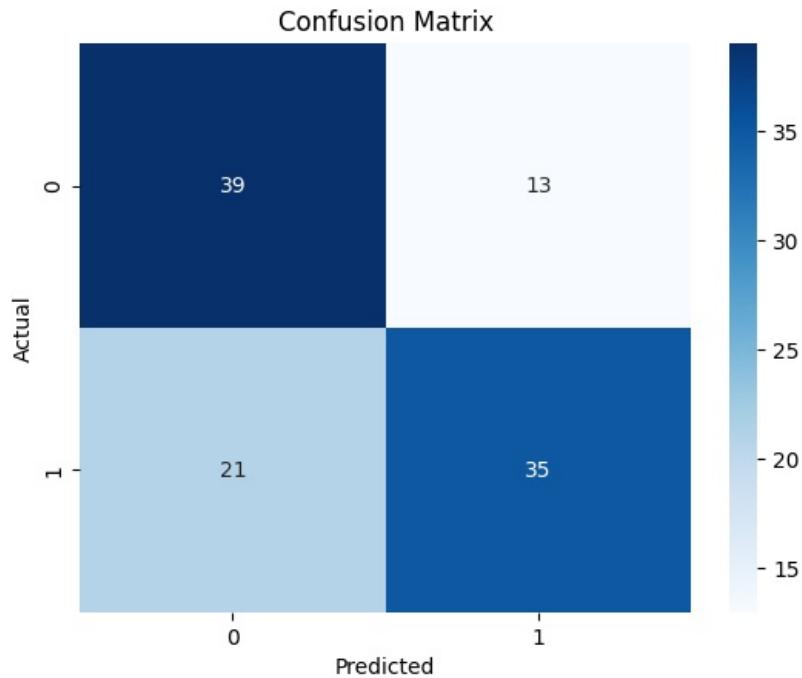
```
In [31]: loss1, accuracy1, precision1, recall1, auc1 = model3_binary.evaluate(X_test, y_test)

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 variation 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)

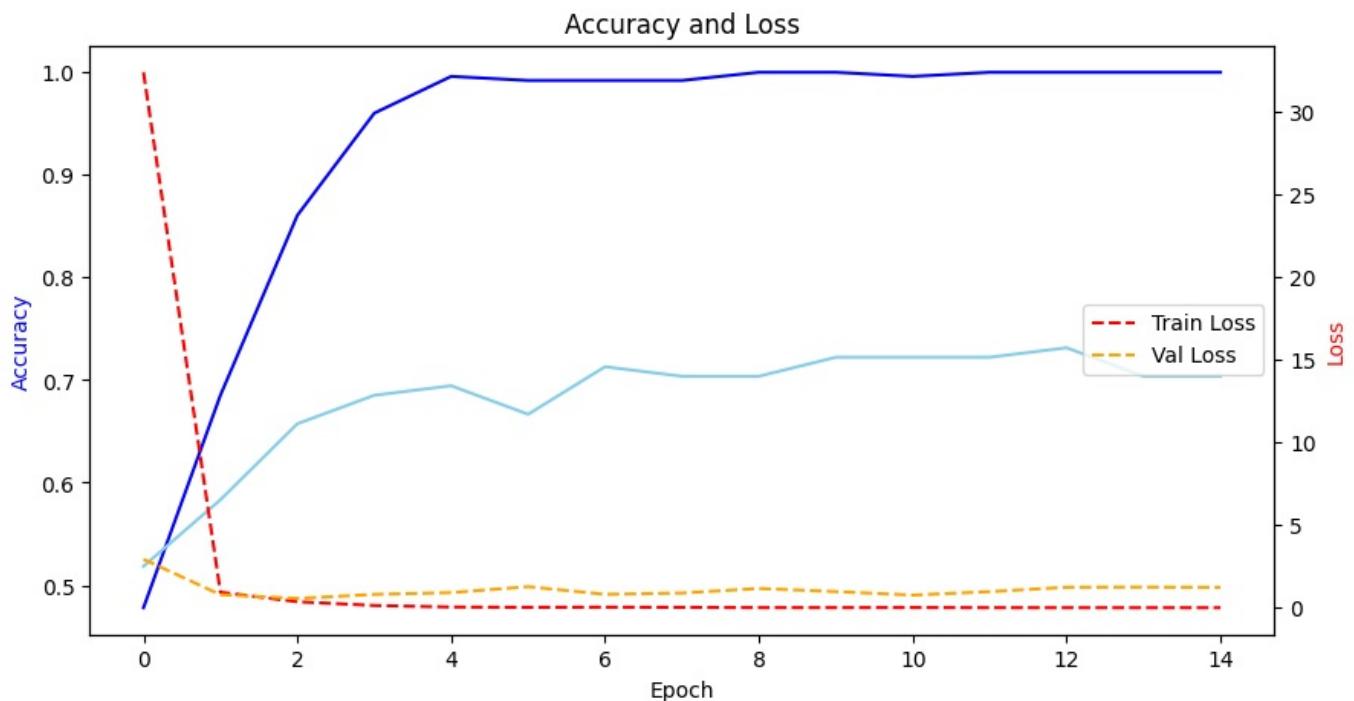
```
{
'model': 'bs16_do4', 'batch_size': 16, 'dropout_rate': 0.4, 'val_accuracy': 0.7037037014961243, 'val_precision': 0.7142857313156128, 'val_recall': 0.7142857313156128, 'val_auc': 0.8027129173278809, 'training_time_sec': 54.82879424095154}
{
'model': 'bs32_do4', 'batch_size': 32, 'dropout_rate': 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}
```

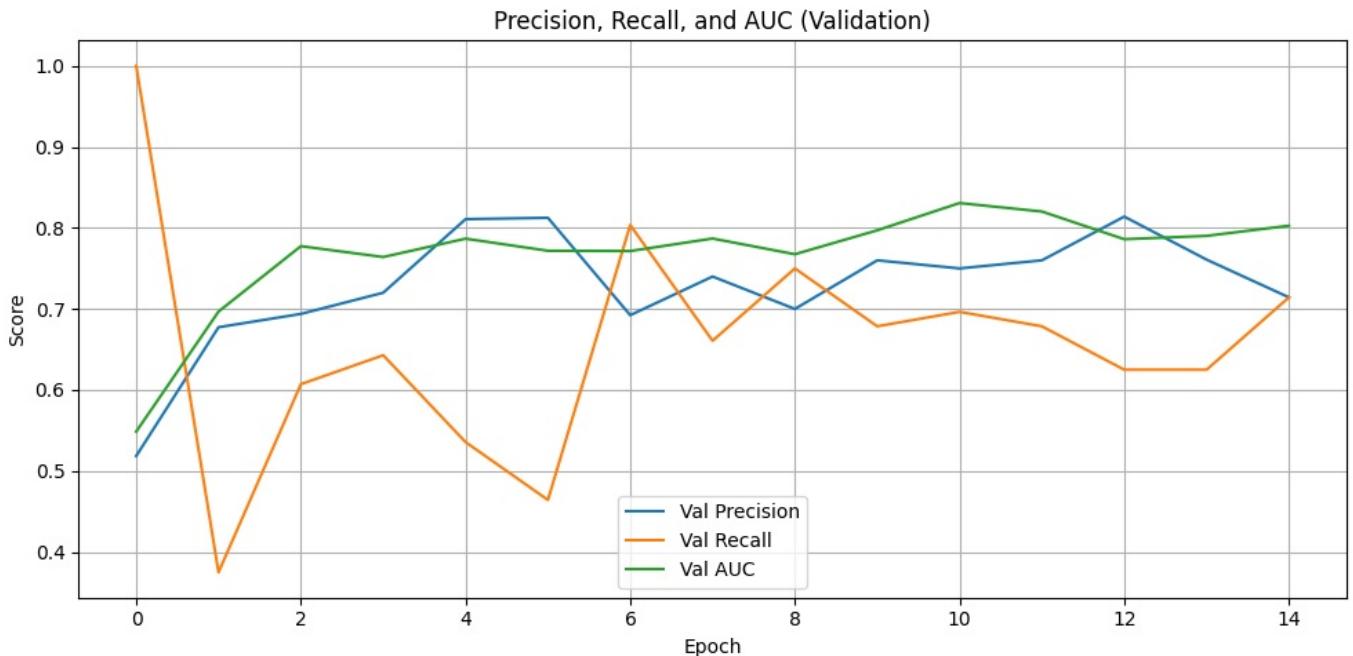
```
In [46]: best_history=history_records1['bs16_do4']
plt.figure(figsize=(10, 5))
plt.title('Accuracy and Loss')
plt.xlabel('Epoch')
```

```
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': 0.722222089767456, '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': 137.50552988052368}
{'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': 0.611111044883728, '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')

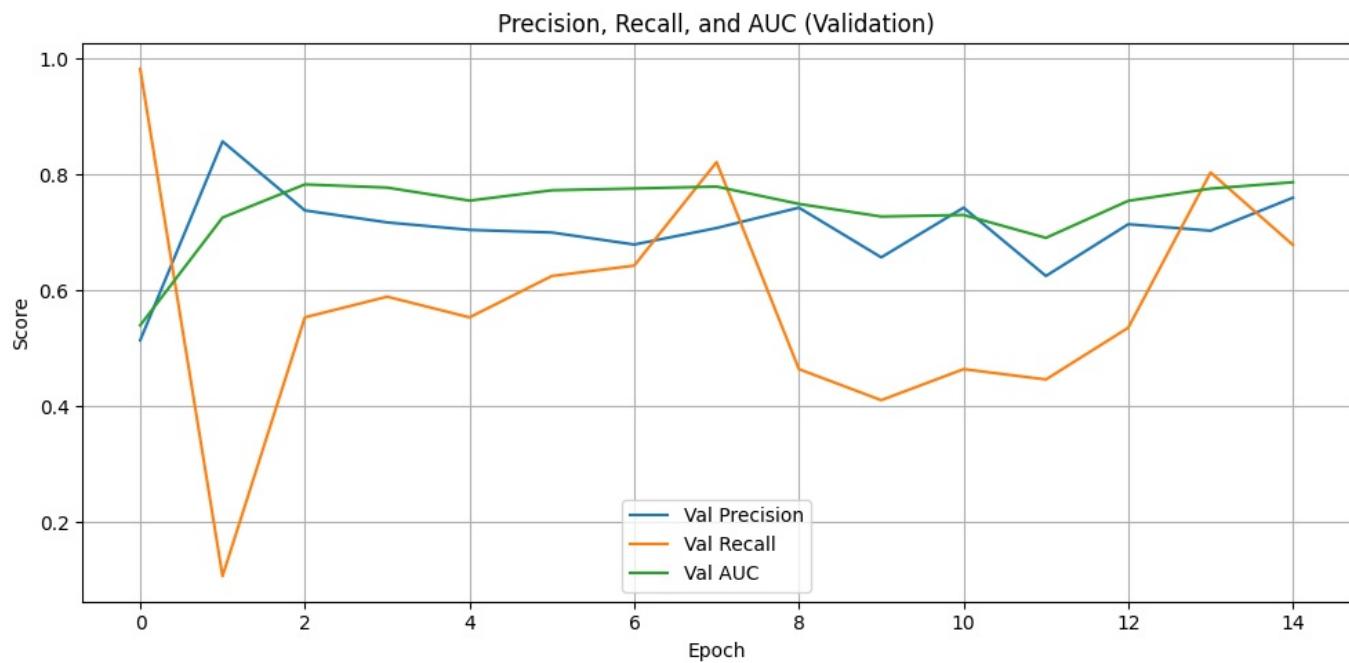
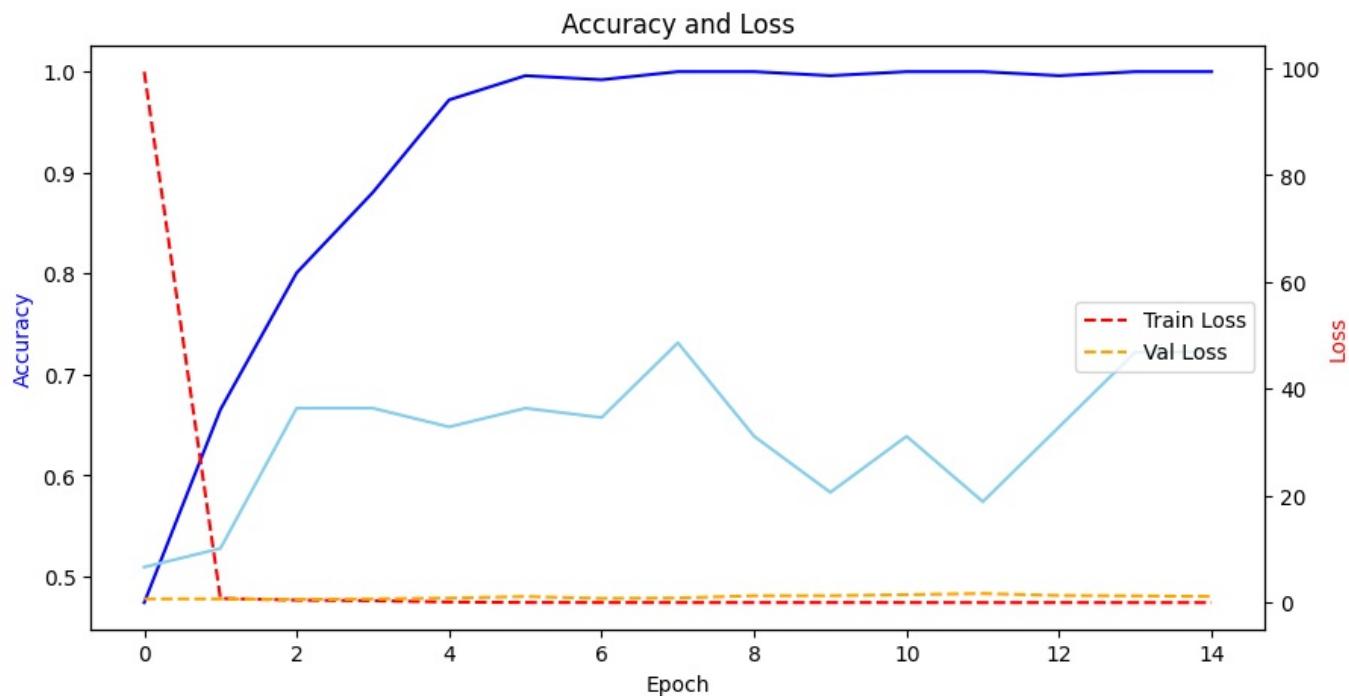
plt.ylabel('Accuracy', color='blue')
plt.twinx()
plt.plot(best_history['loss'], label='Train Loss', color='red', linestyle='--')
plt.plot(best_history['val_loss'], label='Val Loss', color='orange', linestyle='--')
plt.ylabel('Loss', color='red')
plt.legend(loc='center right')
plt.show()

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% accuracy 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\_state.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 [2]: 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 [3]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [4]: # Load the HDF5 file
bird_sound = h5py.File("C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework3\\\\bi
```

```
In [5]: 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 [6]: # Plot one spectrogram sample from each species
unique_species = list(np.unique(y))
sample_specs = {}

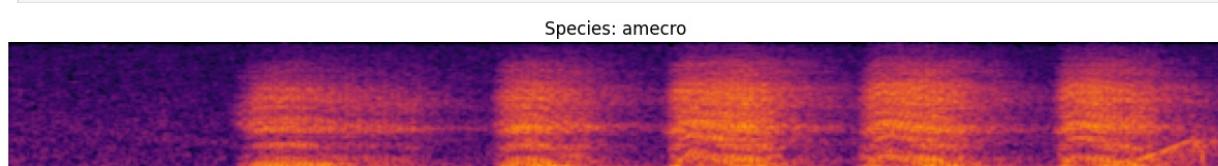
for label in unique_species:
    label_index = y.index(label) if isinstance(y, list) else np.where(np.array(y) == label)[0][0]
    sample = X[label_index, :, :]

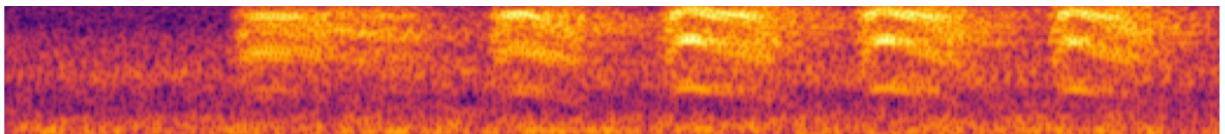
    sample_specs[label] = sample

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

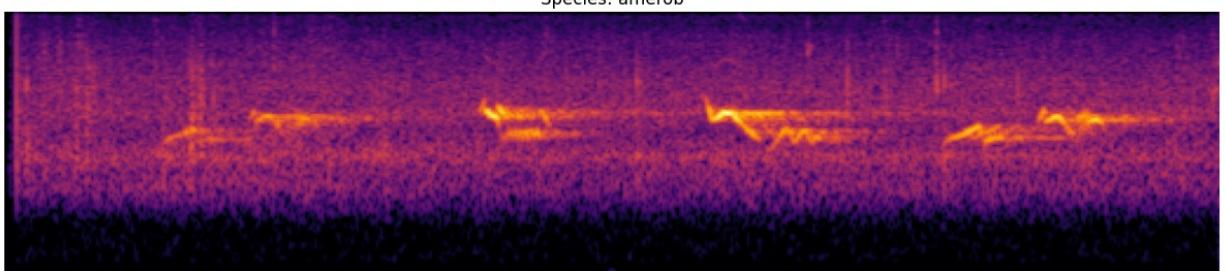
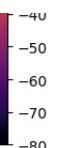
for i, (label, spec) in enumerate(sample_specs.items()):
    plt.subplot(len(sample_specs), 1, i + 1)
    plt.imshow(spec, aspect='auto', origin='lower', cmap='inferno')
    plt.title(f"Species: {label}")
    plt.colorbar()
    plt.axis('off')

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

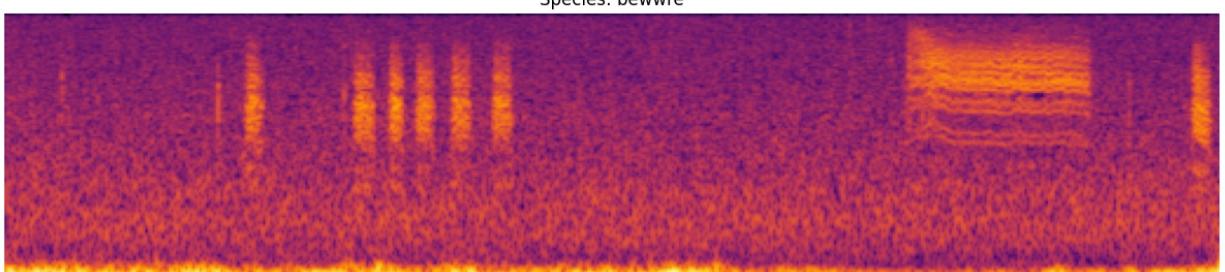
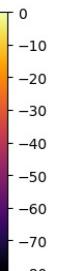




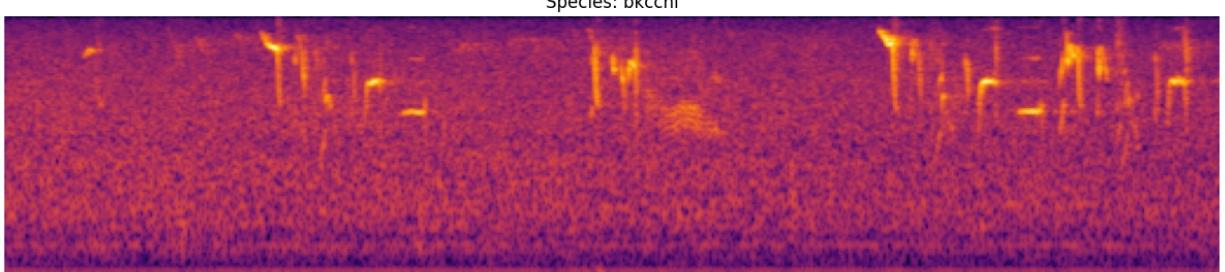
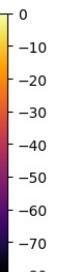
Species: amerob



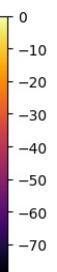
Species: bewwre



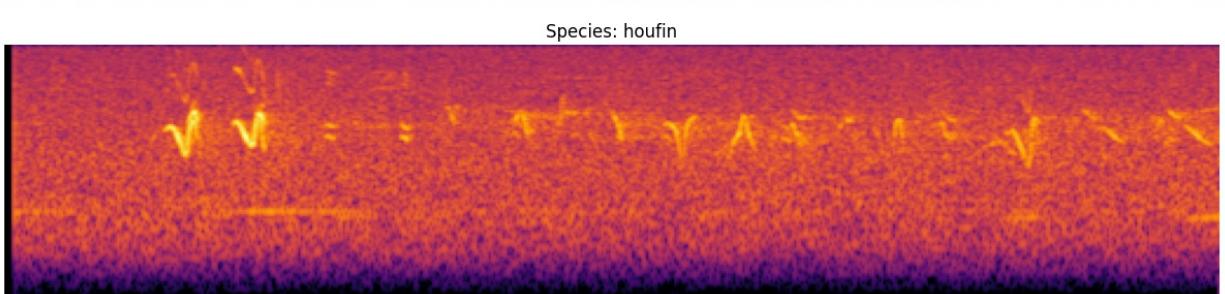
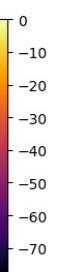
Species: bkcchi



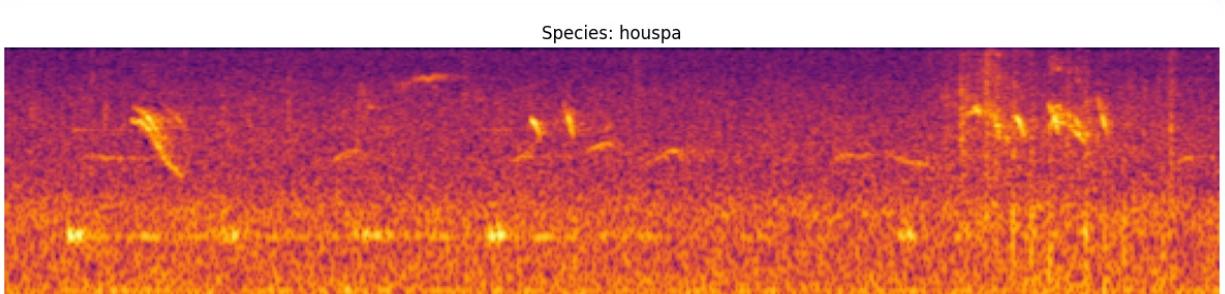
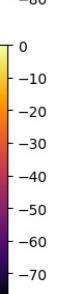
Species: daejun



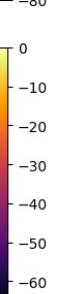
Species: houfin

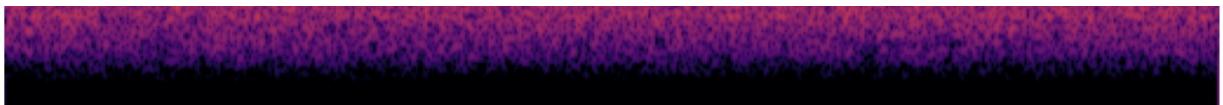


Species: houspa

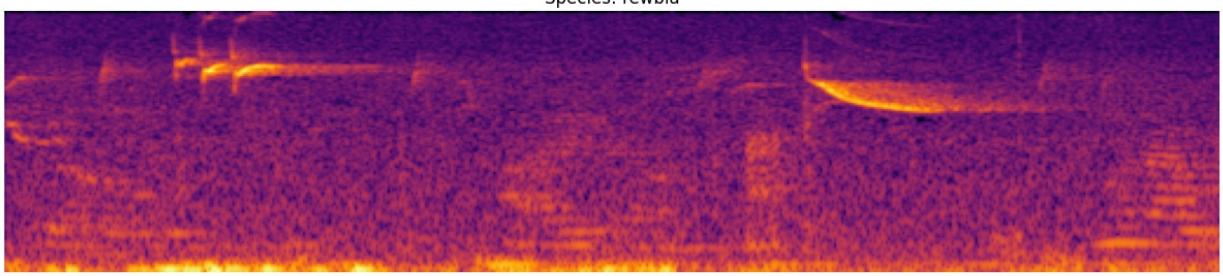


Species: norfli

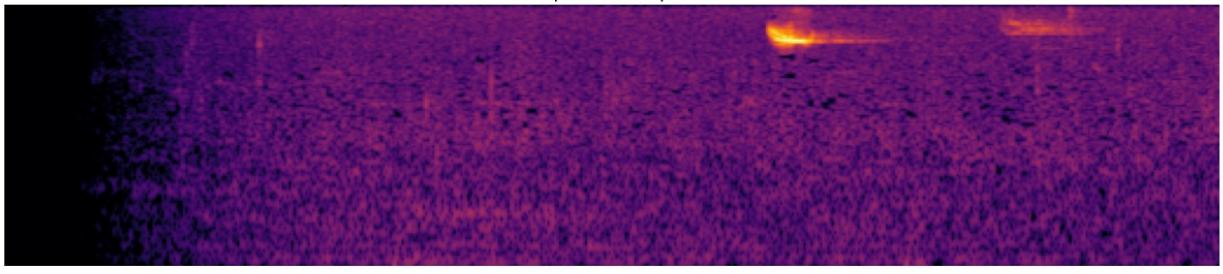
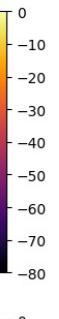




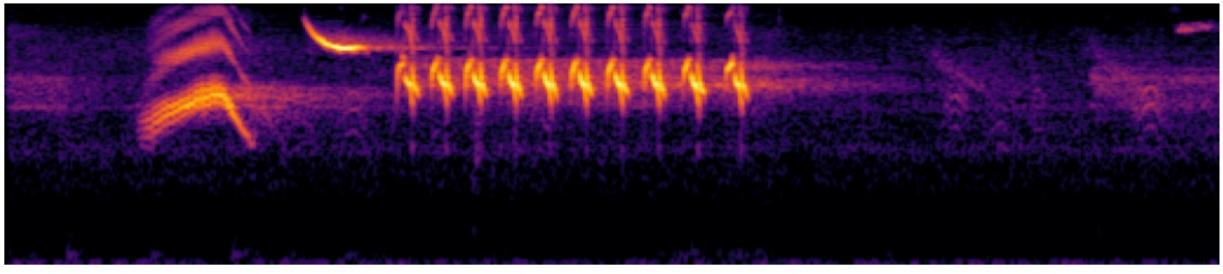
Species: rewbla



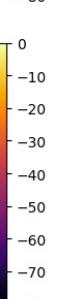
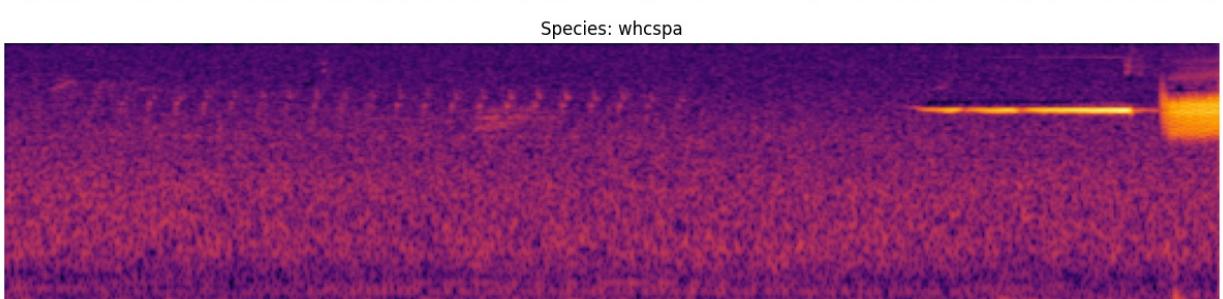
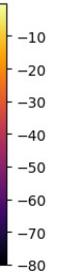
Species: sonspa



Species: spotow



Species: whcspa



## Step2: Train-Test Split

```
In [7]: 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]: # Defining 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: UserWarning:  
 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 0.0182 - val\_accuracy: 0.3602 - val\_auc: 0.7486 - val\_loss: 2.2619 - val\_precision: 1.0000 - val\_recall: 0.0025  
Epoch 6/20  
7/7 32s 5s/step - accuracy: 0.4012 - auc: 0.7967 - loss: 2.0732 - precision: 0.8099 - recall: 0.0506 - val\_accuracy: 0.4030 - val\_auc: 0.7713 - val\_loss: 2.0368 - val\_precision: 0.5932 - val\_recall: 0.0882  
Epoch 7/20  
7/7 32s 5s/step - accuracy: 0.4880 - auc: 0.8424 - loss: 1.7971 - precision: 0.6669 - recall: 0.1303 - val\_accuracy: 0.4081 - val\_auc: 0.7851 - val\_loss: 2.0887 - val\_precision: 0.5025 - val\_recall: 0.2569  
Epoch 8/20  
7/7 32s 4s/step - accuracy: 0.5516 - auc: 0.8821 - loss: 1.5779 - precision: 0.7027 - recall: 0.2674 - val\_accuracy: 0.4131 - val\_auc: 0.7977 - val\_loss: 1.9996 - val\_precision: 0.5183 - val\_recall: 0.2141  
Epoch 9/20  
7/7 33s 5s/step - accuracy: 0.6463 - auc: 0.9291 - loss: 1.2091 - precision: 0.7768 - recall: 0.4323 - val\_accuracy: 0.3703 - val\_auc: 0.7754 - val\_loss: 2.2326 - val\_precision: 0.4770 - val\_recall: 0.2091  
Epoch 10/20  
7/7 34s 5s/step - accuracy: 0.7044 - auc: 0.9524 - loss: 1.0085 - precision: 0.8508 - recall: 0.4959 - val\_accuracy: 0.4257 - val\_auc: 0.8066 - val\_loss: 2.0428 - val\_precision: 0.5694 - val\_recall: 0.2997  
Epoch 11/20  
7/7 33s 5s/step - accuracy: 0.7876 - auc: 0.9773 - loss: 0.7180 - precision: 0.9011 - recall: 0.6416 - val\_accuracy: 0.4408 - val\_auc: 0.8011 - val\_loss: 2.0789 - val\_precision: 0.5631 - val\_recall: 0.2922  
Epoch 12/20  
7/7 32s 5s/step - accuracy: 0.8431 - auc: 0.9899 - loss: 0.5213 - precision: 0.9326 - recall: 0.7390 - val\_accuracy: 0.4332 - val\_auc: 0.8012 - val\_loss: 2.1565 - val\_precision: 0.5402 - val\_recall: 0.3048  
Epoch 13/20  
7/7 32s 5s/step - accuracy: 0.9085 - auc: 0.9956 - loss: 0.3503 - precision: 0.9638 - recall: 0.8365 - val\_accuracy: 0.4559 - val\_auc: 0.7993 - val\_loss: 2.3629 - val\_precision: 0.5349 - val\_recall: 0.3476  
Epoch 14/20  
7/7 31s 4s/step - accuracy: 0.9285 - auc: 0.9976 - loss: 0.2582 - precision: 0.9752 - recall: 0.8818 - val\_accuracy: 0.4307 - val\_auc: 0.8021 - val\_loss: 2.4970 - val\_precision: 0.5068 - val\_recall: 0.3778  
Epoch 15/20  
7/7 32s 5s/step - accuracy: 0.9570 - auc: 0.9992 - loss: 0.1659 - precision: 0.9755 - recall: 0.9340 - val\_accuracy: 0.4408 - val\_auc: 0.8051 - val\_loss: 2.5254 - val\_precision: 0.4765 - val\_recall: 0.3577  
Epoch 16/20  
7/7 31s 4s/step - accuracy: 0.9673 - auc: 0.9994 - loss: 0.1530 - precision: 0.9851 - recall: 0.9394 - val\_accuracy: 0.4257 - val\_auc: 0.7952 - val\_loss: 2.8406 - val\_precision: 0.4763 - val\_recall: 0.3804  
Epoch 17/20  
7/7 31s 4s/step - accuracy: 0.9562 - auc: 0.9993 - loss: 0.1310 - precision: 0.9726 - recall: 0.9422 - val\_accuracy: 0.4710 - val\_auc: 0.8041 - val\_loss: 2.5447 - val\_precision: 0.5288 - val\_recall: 0.3929  
Epoch 18/20  
7/7 31s 4s/step - accuracy: 0.9813 - auc: 0.9996 - loss: 0.0987 - precision: 0.9884 - recall: 0.9640 - val\_accuracy: 0.4332 - val\_auc: 0.8000 - val\_loss: 2.6519 - val\_precision: 0.4901 - val\_recall: 0.3753  
Epoch 19/20  
7/7 31s 4s/step - accuracy: 0.9797 - auc: 0.9997 - loss: 0.0756 - precision: 0.9891 - recall: 0.9743 - val\_accuracy: 0.4257 - val\_auc: 0.7952 - val\_loss: 2.9259 - val\_precision: 0.4985 - val\_recall: 0.4106  
Epoch 20/20  
7/7 31s 4s/step - accuracy: 0.9854 - auc: 0.9999 - loss: 0.0571 - precision: 0.9896 - recall: 0.9803 - val\_accuracy: 0.4332 - val\_auc: 0.7920 - val\_loss: 2.7538 - val\_precision: 0.4903 - val\_recall: 0.3829

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
```

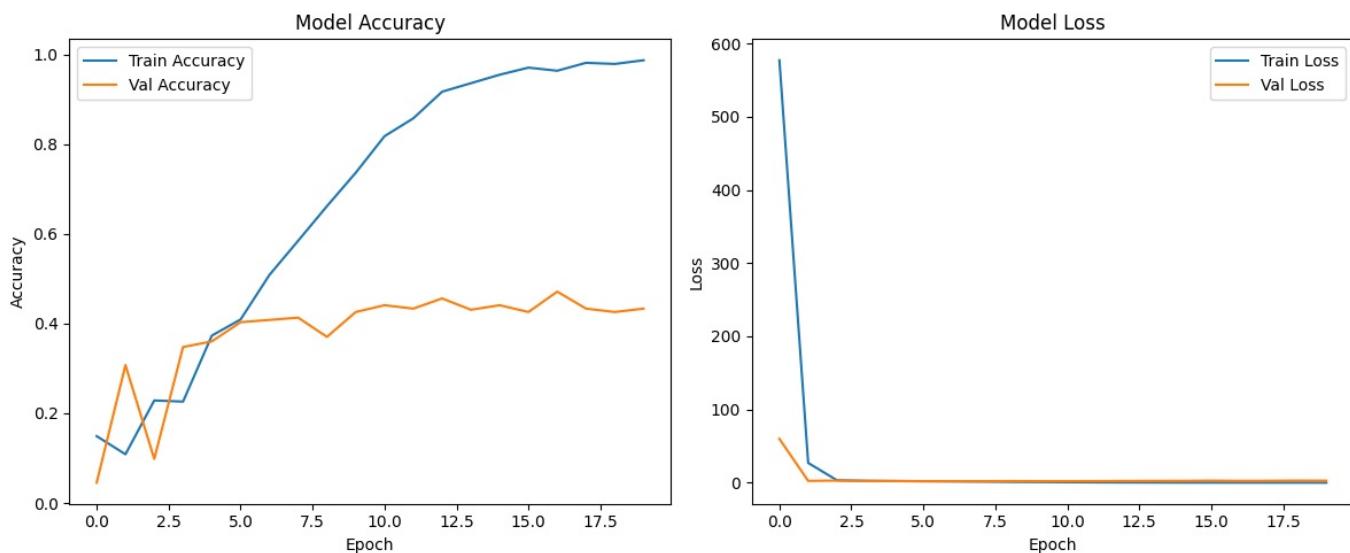
- Model 1 has validation accuracy of (43.32%) and a strong AUC of 0.7844. These results indicate effective overall classification across multiple bird species.

```
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

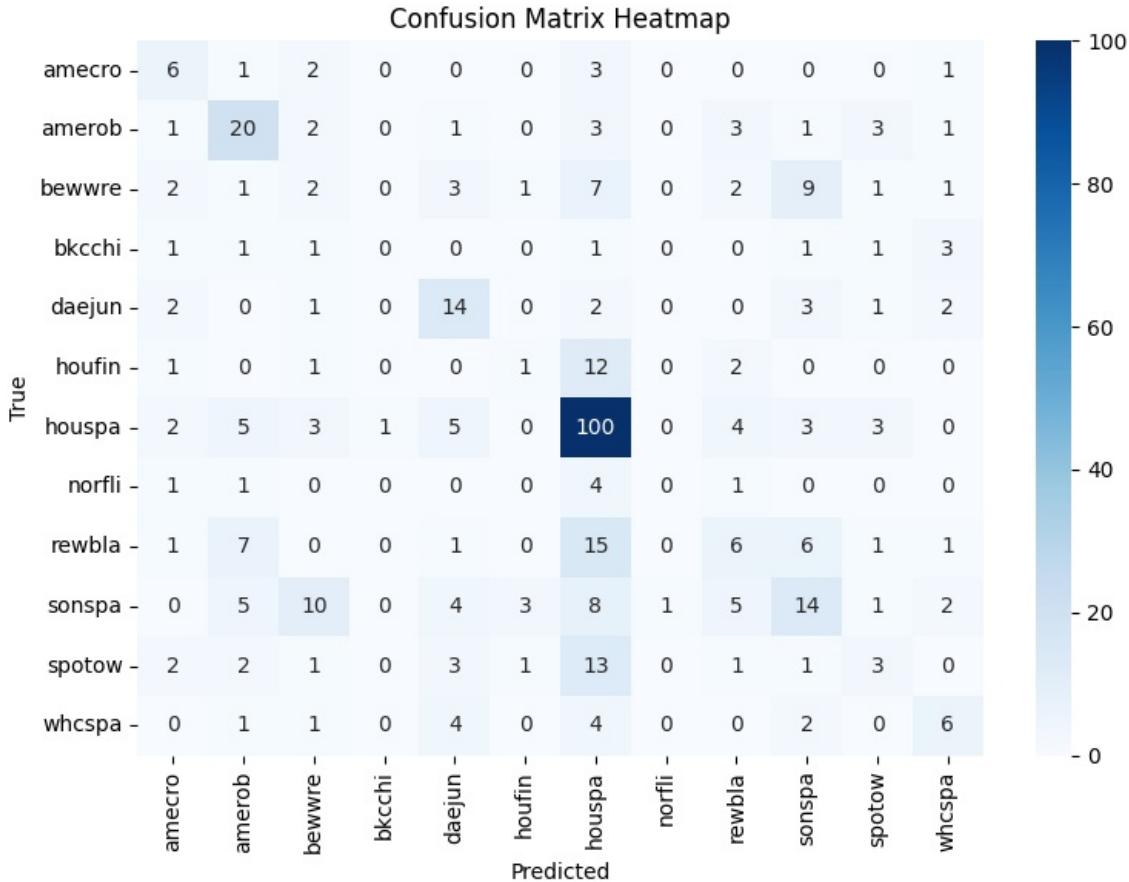
y_pred = model1_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

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_recall: 0.0000e+00
Epoch 2/15
13/13 ━━━━━━ 9s 654ms/step - accuracy: 0.1788 - auc: 0.6281 - loss: 2.4847 - precision: 0.0977 - recall: 0.0225 - val_accuracy: 0.2771 - val_auc: 0.6823 - val_loss: 2.3905 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/15
13/13 ━━━━ 9s 670ms/step - accuracy: 0.2556 - auc: 0.6828 - loss: 2.4334 - precision: 0.3094 - recall: 0.0339 - val_accuracy: 0.3149 - val_auc: 0.7282 - val_loss: 2.2707 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 4/15
13/13 ━━━━ 9s 681ms/step - accuracy: 0.3738 - auc: 0.7624 - loss: 2.1663 - precision: 0.5728 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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
```

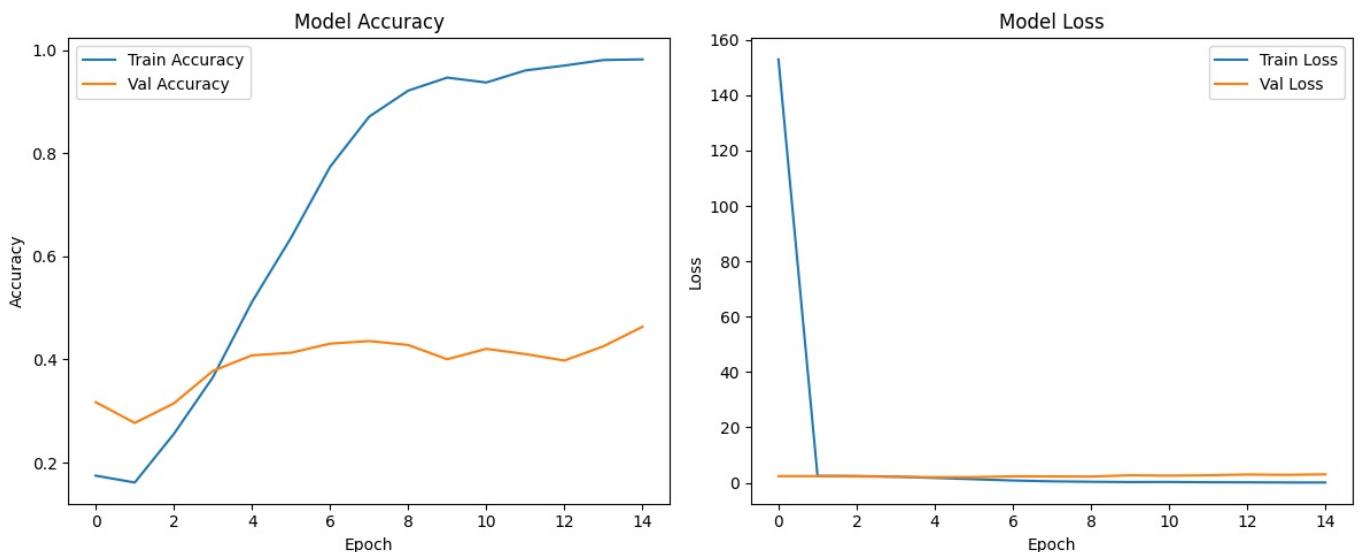
- Model 2 has validation accuracy of (46.35%) and a strong AUC of 0.7810. These results shows that it has 3% moere accuracy than model1.

```
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()
```



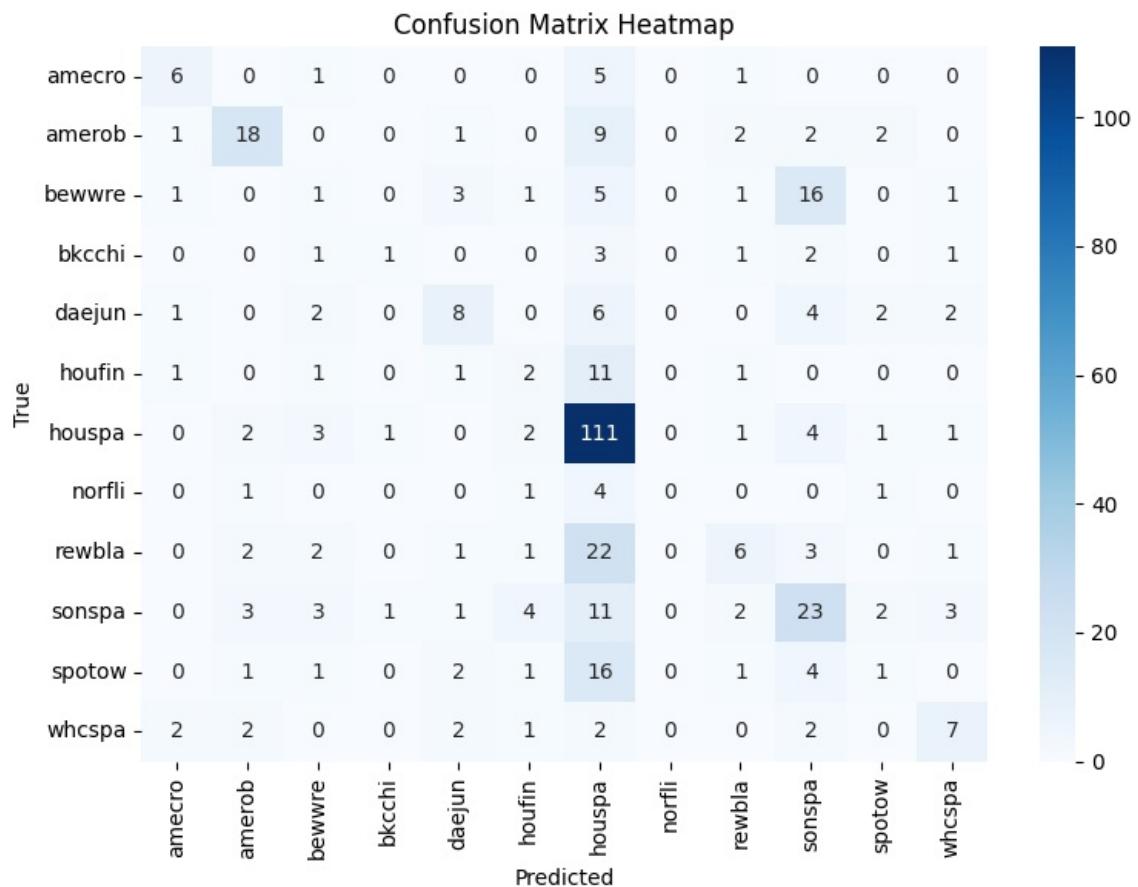
```
In [17]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

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: UserWarning
  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]: 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 - recall: 0.1436 - val\_accuracy: 0.0227 - val\_auc: 0.5114 - val\_loss: 2.4803 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00  
Epoch 2/20  
13/13 33s 3s/step - accuracy: 0.1077 - auc: 0.5389 - loss: 4.6898 - precision: 0.1610 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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}")

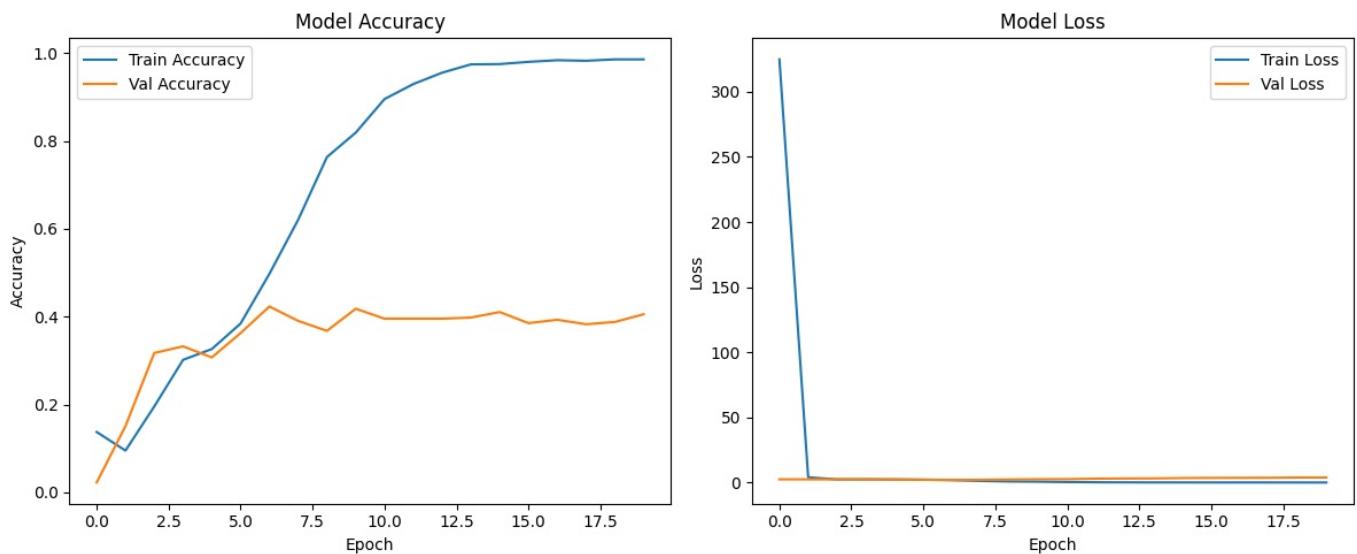
Validation Accuracy Model3:
val_accuracy: 0.4055
val_precision: 0.4332
val_recall: 0.3678
val_auc: 0.7673
```

```
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()
```

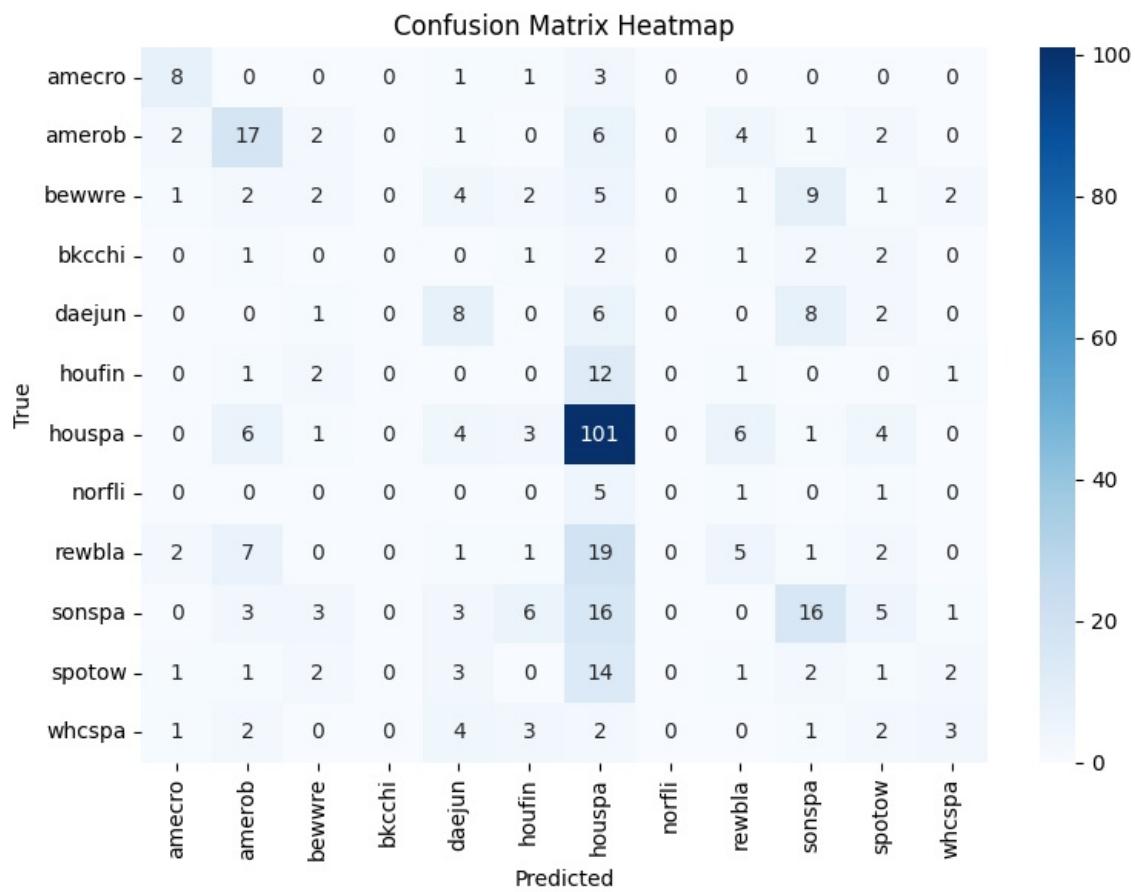


```
In [22]: y_pred = model1a_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

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 - recall: 0.1203 - val_accuracy: 0.3652 - val_auc: 0.7518 - val_loss: 2.2955 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 2/15
25/25 ━━━━━━━━━━ 34s 1s/step - accuracy: 0.3851 - auc: 0.7703 - loss: 2.1832 - precision: 0.5867 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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 - recall: 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:

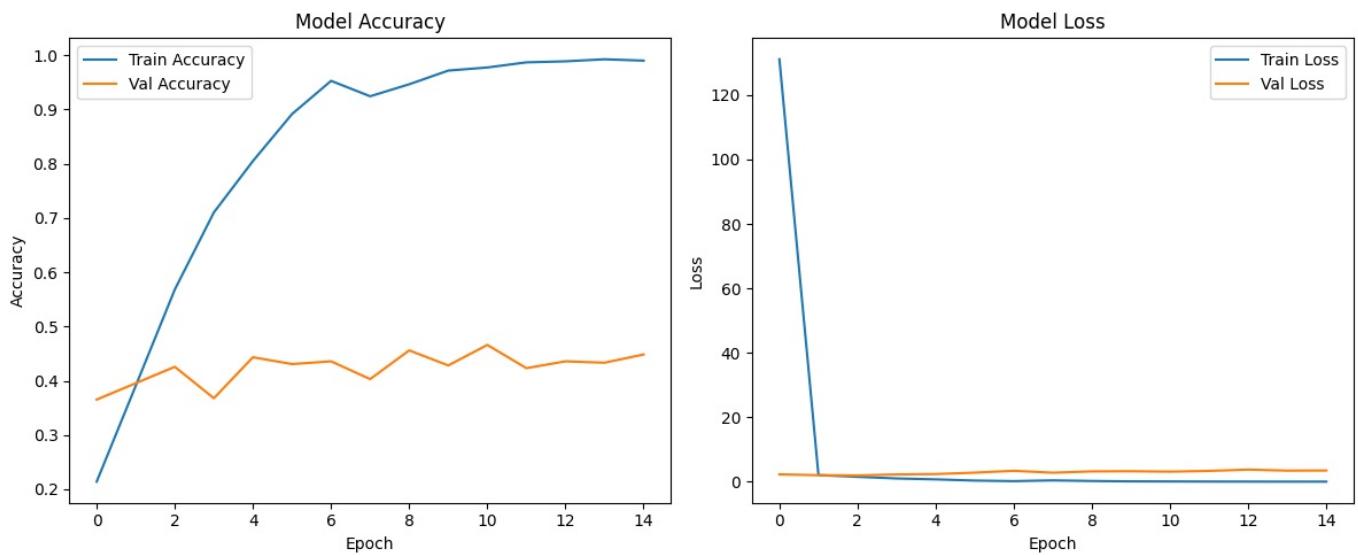
val\_accuracy: 0.4484  
 val\_precision: 0.4926  
 val\_recall: 0.4207  
 val\_auc: 0.7943

```
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()
```

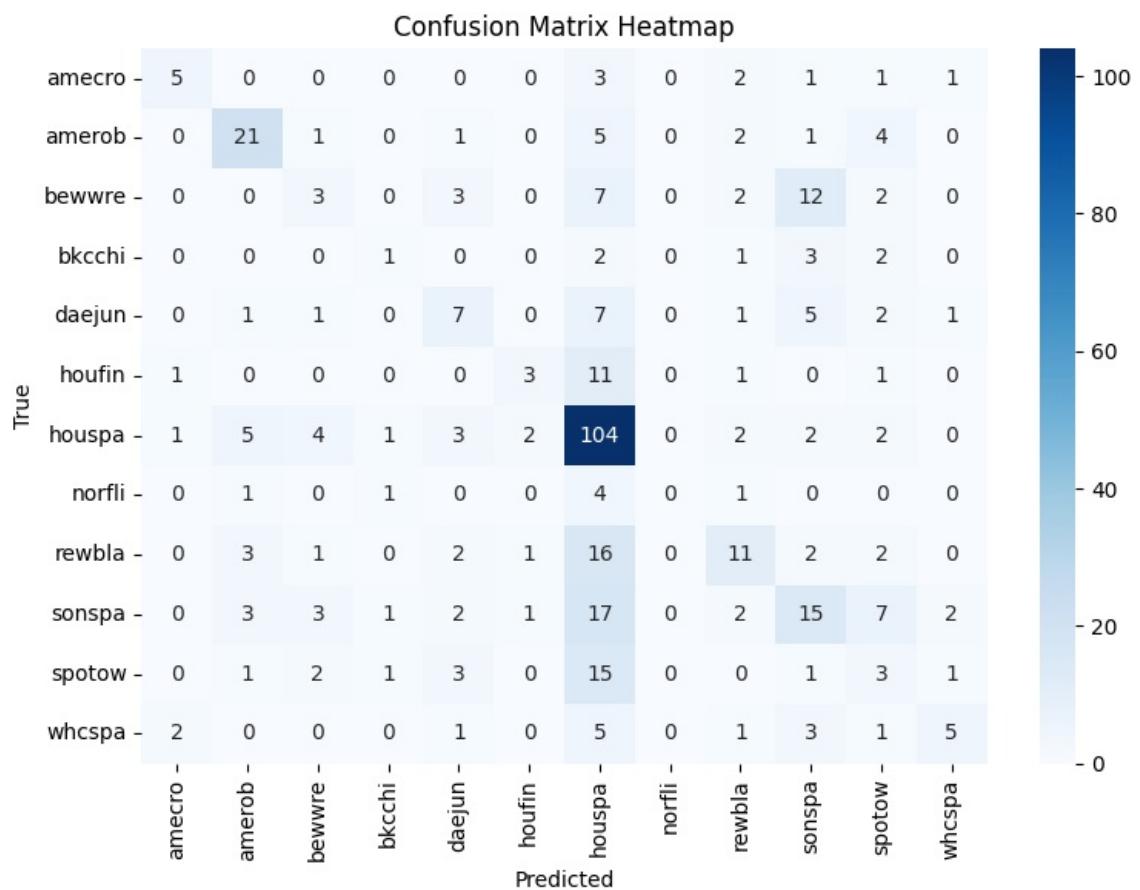


```
In [27]: y_pred = model1b_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

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\_recall: 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()

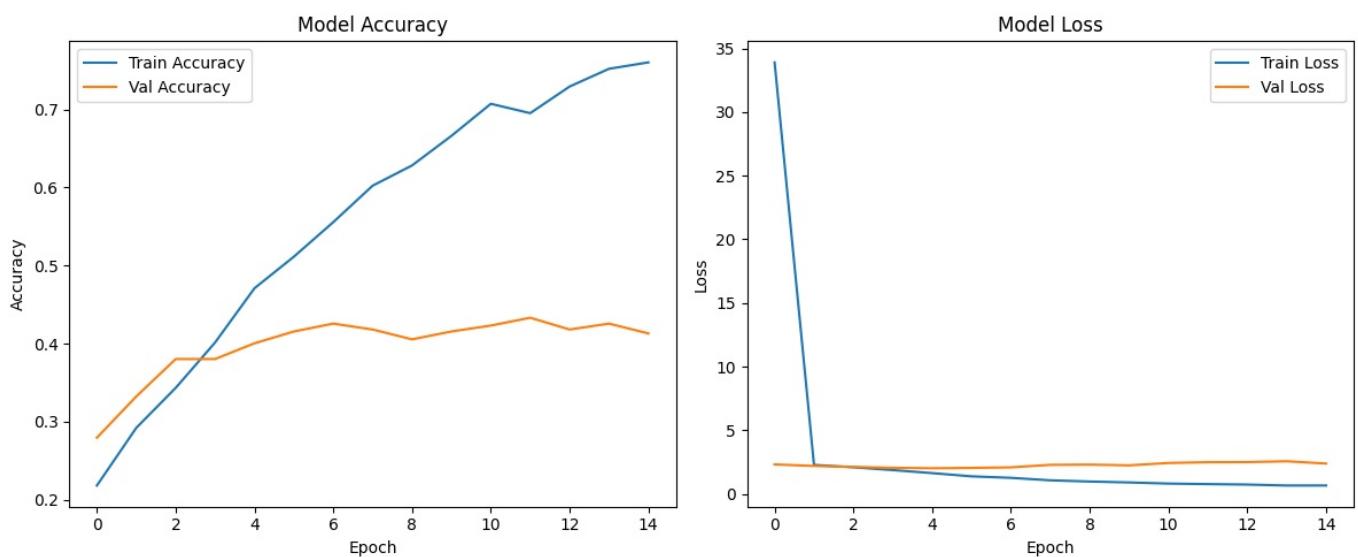
plt.subplot(1, 2, 2)
plt.plot(history6.history['loss'], label='Train Loss')
```

```

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()

```



```

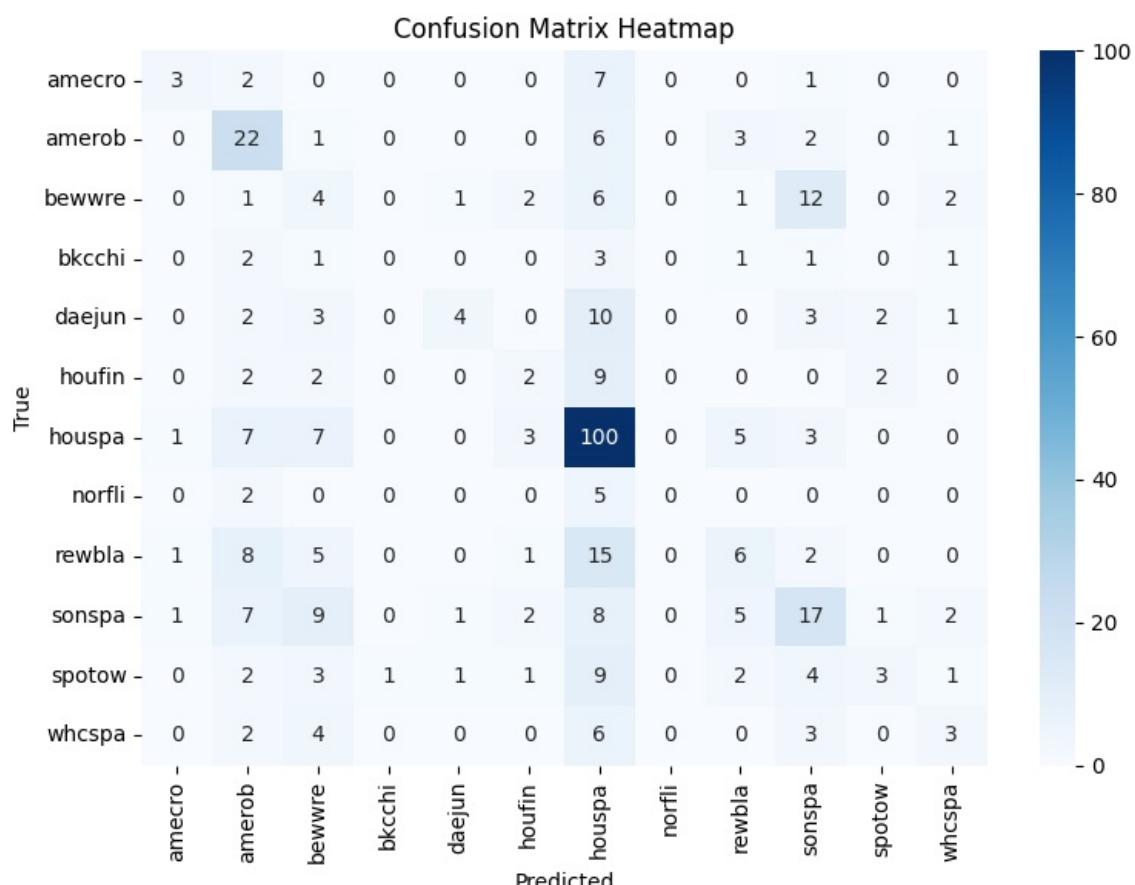
In [32]: y_pred = model2a_multi.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

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



```
In [33]: np.random.seed(160)
tf.random.set_seed(160)
```

```
model2b_multi = Sequential()
model2b_multi.add(Conv2D(16, (3, 3), activation='relu', input_shape=(128, 517, 1)))
model2b_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2b_multi.add(Conv2D(32, (3, 3), activation='relu'))
model2b_multi.add(MaxPooling2D(pool_size=(2, 2)))

model2b_multi.add(Flatten())
model2b_multi.add(Dense(64, activation='relu'))
model2b_multi.add(Dropout(0.4))
```

```

model2b_multi.add(Dense(12, activation='softmax'))

model2b_multi.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        Precision(name='precision'),
        Recall(name='recall'),
        AUC(name='auc')
    ]
)

model2b_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\_5"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 126, 515, 16)	160
max_pooling2d_10 (MaxPooling2D)	(None, 63, 257, 16)	0
conv2d_11 (Conv2D)	(None, 61, 255, 32)	4,640
max_pooling2d_11 (MaxPooling2D)	(None, 30, 127, 32)	0
flatten_5 (Flatten)	(None, 121920)	0
dense_10 (Dense)	(None, 64)	7,802,944
dropout_5 (Dropout)	(None, 64)	0
dense_11 (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 [46]: `# saving bestmodel out of all  
model1b_multi.save("C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework3\\\\bestmodel.h5")`

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.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\\\\bestmodel.h5")`

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.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 [61]: `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\\\\Homework 3\\\\Model 1")
model2 = tf.keras.models.load_model("C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework 3\\\\Model 2")
models = {"Model 1": model1, "Model 2": model2}

mp3_files = [
    "C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework3\\\\Homework 3_test_birds_1.mp3",
    "C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework3\\\\Homework 3_test_birds_2.mp3",
    "C:\\\\Users\\\\Nidhi\\\\Documents\\\\College Material\\\\Spring Quarter 2025\\\\SML2\\\\Homework3\\\\Homework 3_test_birds_3.mp3"
]

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` will 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` will be empty until you train or evaluate the model.

```

In [62]: for model_idx, model in enumerate([model1, model2], start=1):
    print(f"\n{='-'*30}")
    print(f"Predictions using Model {model_idx}")
    print(f"{'='*30}")

    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()

```

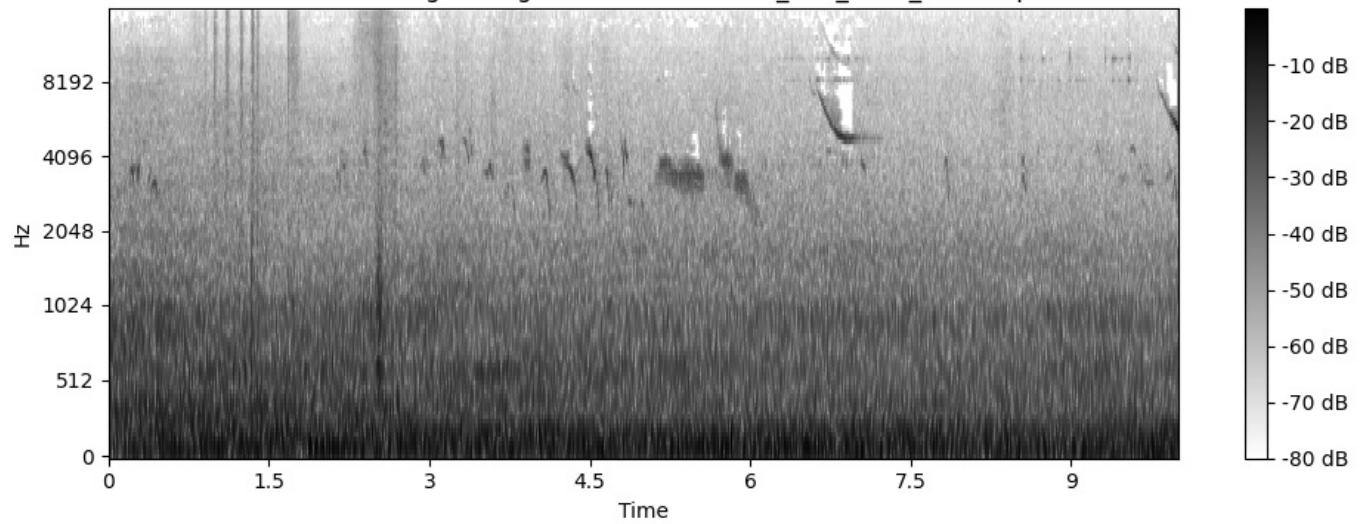
=====  
Predictions using Model 1  
=====

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

Model 1 - Strongest Segment - Homework 3\_test\_birds\_test1.mp3

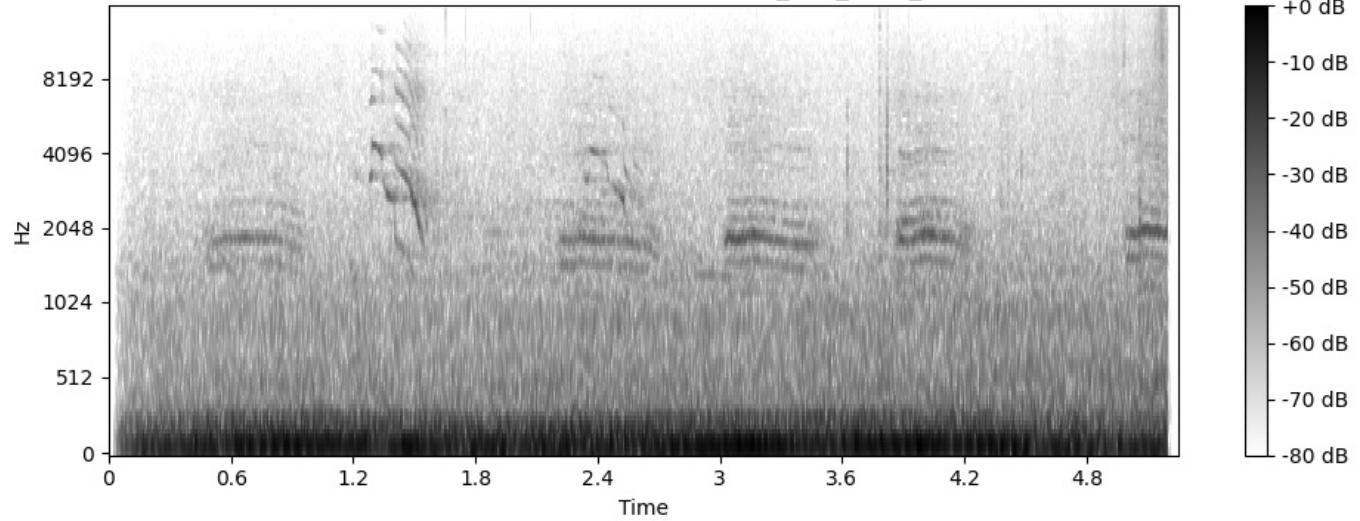


File: Homework 3\_test\_birds\_test2.mp3  
Top 3 Average Predictions:

- rewbla: 0.59
- daejun: 0.16
- sonspa: 0.14

Top species per window:  
- rewbla: 1 windows

Model 1 - Strongest Segment - Homework 3\_test\_birds\_test2.mp3



File: Homework 3\_test\_birds\_test3.mp3

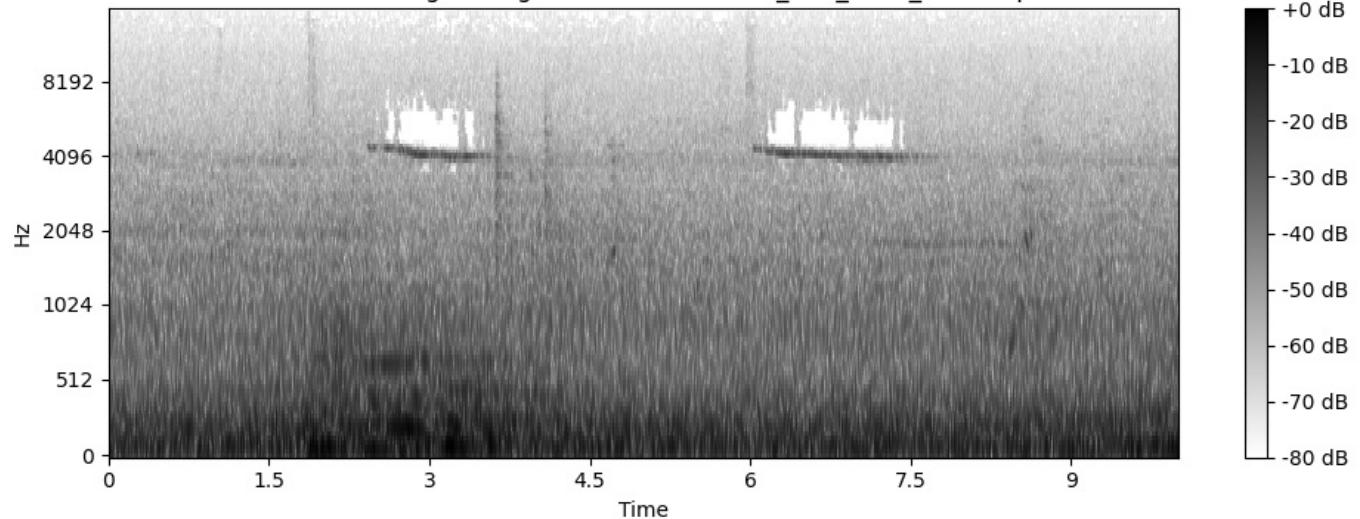
Top 3 Average Predictions:

- rewbla: 0.92
- daejun: 0.04
- sonspa: 0.03

Top species per window:

- rewbla: 4 windows

Model 1 - Strongest Segment - Homework 3\_test\_birds\_test3.mp3



=====

Predictions using Model 2

=====

File: Homework 3\_test\_birds\_test1.mp3

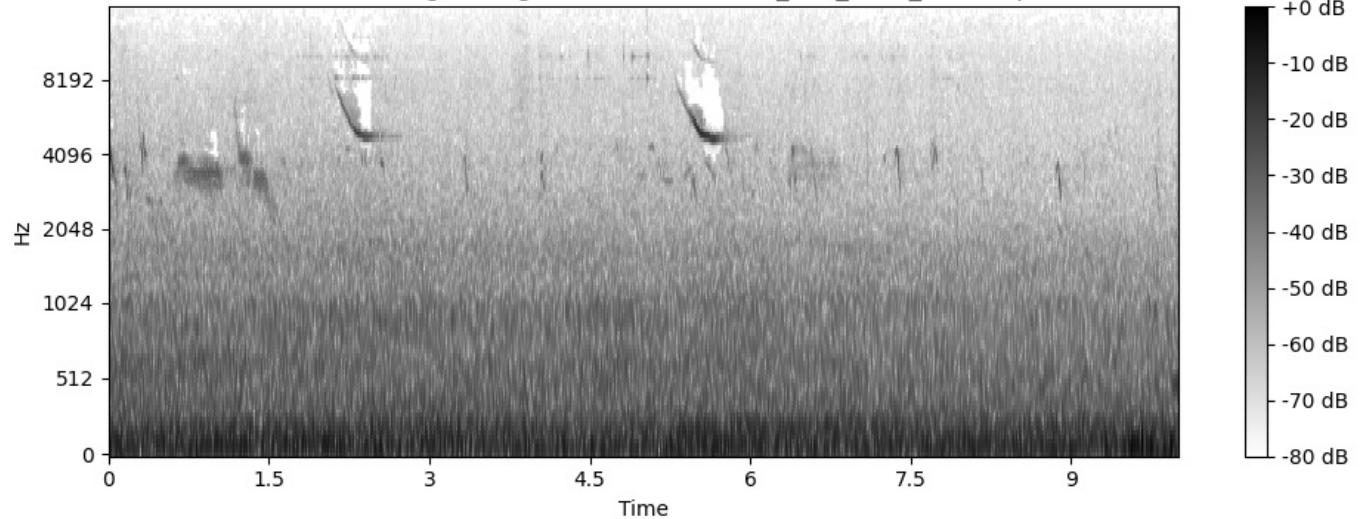
Top 3 Average Predictions:

- houspa: 0.45
- sonspa: 0.28
- houfin: 0.27

Top species per window:

- houspa: 9 windows

Model 2 - Strongest Segment - Homework 3\_test\_birds\_test1.mp3



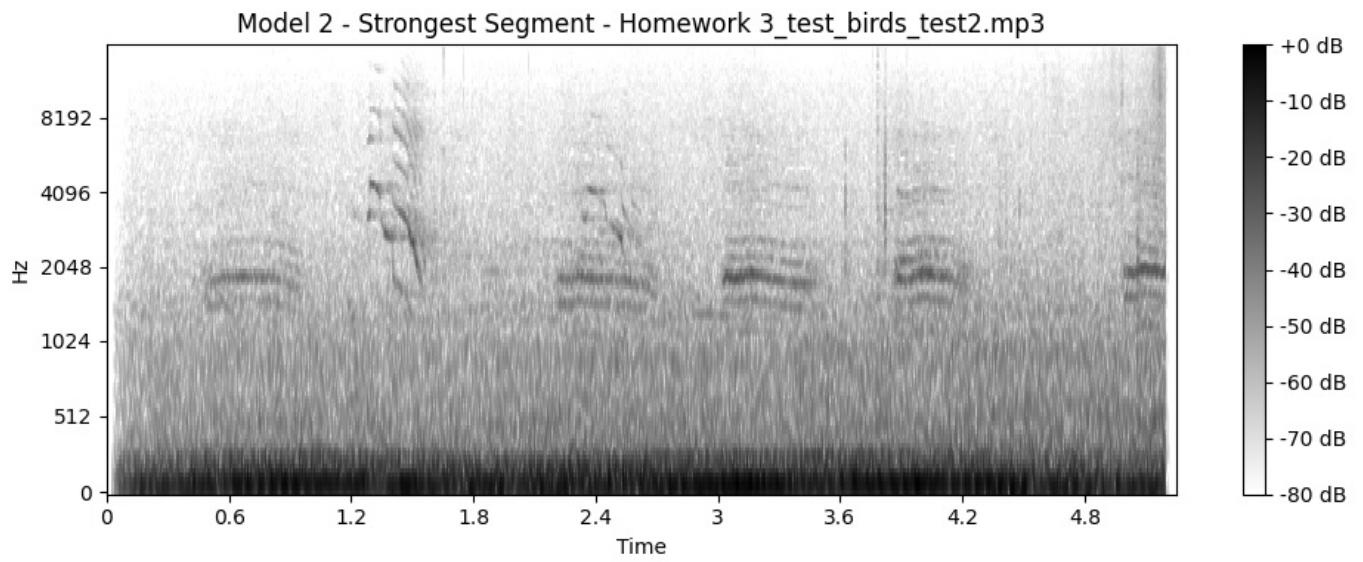
File: Homework 3\_test\_birds\_test2.mp3

Top 3 Average Predictions:

- houspa: 0.42
- sonspa: 0.29
- houfin: 0.28

Top species per window:

- houspa: 1 windows



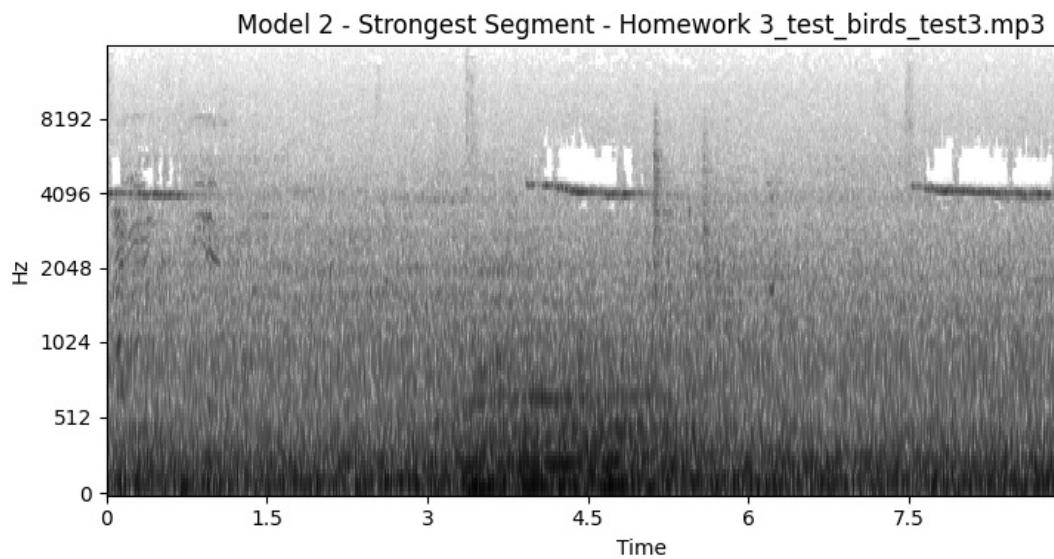
File: Homework 3\_test\_birds\_test3.mp3

Top 3 Average Predictions:

- houspa: 0.48
- sonspa: 0.27
- houfin: 0.25

Top species per window:

- houspa: 4 windows



In [ ]: