Copy_All_New_dtest_3

May 12, 2025

1 Predicting Sounds of Seattle Birds

1.1 Hrishabh Kulkarni

2 1. Binary Model

Pick two bird species and build a network to do binary classification on these two alone.

[1]:
```
import time
import numpy as np
import h5py
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle, class_weight
from sklearn.metrics import (accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, roc_curve, confusion_matrix, ConfusionMatrixDisplay)
from sklearn.decomposition import PCA
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam, RMSprop
from keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[2]:
```
SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)

# 1. Species selection and data loading
sp1_code, sp2_code = 'whcspa', 'rewbla'
species_map = {
    'whcspa': 'White-crowned Sparrow',
    'rewbla': 'Red-winged Blackbird'
}

with h5py.File('bird_spectrograms.hdf5','r') as f:
```

```
    d1 = f[sp1_code][:]
    d2 = f[sp2_code][:]
```

[3]:
```
# 2. Data inspection, validation, preprocessing
print(f"\nData shapes - {species_map[sp1_code]}: {d1.shape},␣
 ↪{species_map[sp2_code]}: {d2.shape}")
print(f"Sample values - Min: {min(d1.min(), d2.min())}, Max: {max(d1.max(), d2.
 ↪max())}")

min_t = min(d1.shape[2], d2.shape[2])

def normalize_spectrograms(data):
    return (data - np.mean(data)) / (np.std(data) + 1e-8)

d1 = normalize_spectrograms(d1[:,:,:min_t])
d2 = normalize_spectrograms(d2[:,:,:min_t])

X = np.concatenate([d1, d2], axis=0)[..., None]
y = np.concatenate([np.zeros(len(d1)), np.ones(len(d2))])

# Verifying class balance
print(f"\nClass distribution - {species_map[sp1_code]}: {len(d1)},␣
 ↪{species_map[sp2_code]}: {len(d2)}")
```

```
Data shapes - White-crowned Sparrow: (128, 517, 91), Red-winged Blackbird: (128,
517, 187)
Sample values - Min: -80.0, Max: 1.9073486328125e-06

Class distribution - White-crowned Sparrow: 128, Red-winged Blackbird: 128
```

[4]:
```
# 3. train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.24,
    stratify=y,
    random_state=SEED
)

print(f"\nTrain shape: {X_train.shape}, Test shape: {X_test.shape}")

# 4. Class weights for imbalanced data
classes = np.unique(y_train)
cw_vals = class_weight.compute_class_weight('balanced', classes=classes,␣
 ↪y=y_train)
cw = {int(c): w for c, w in zip(classes, cw_vals)}
print(f"\nClass weights: {cw}")
```

```
# 5. Now, implementing Data augmentation
datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    validation_split=0.2
)
```

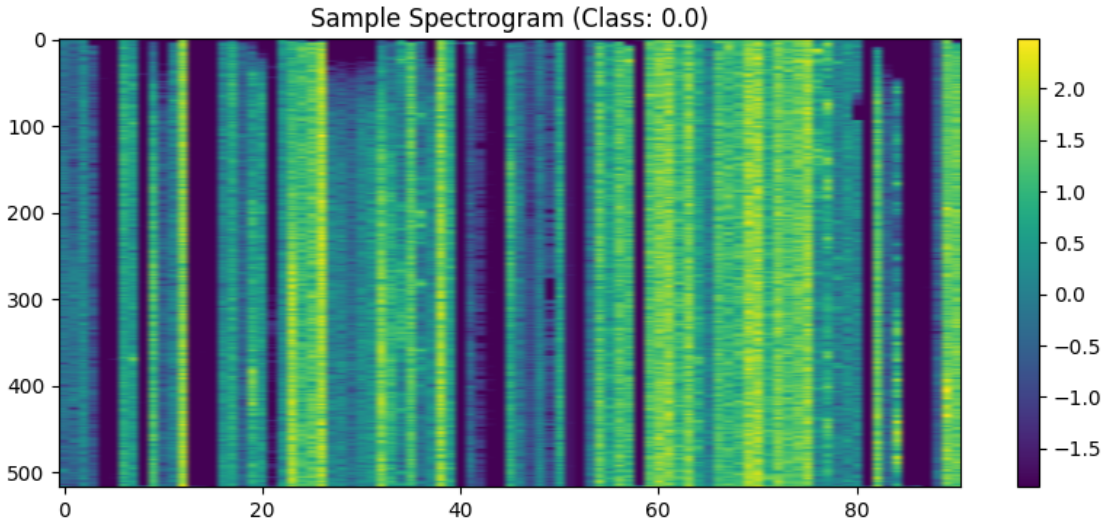
```
Train shape: (194, 517, 91, 1), Test shape: (62, 517, 91, 1)

Class weights: {0: np.float64(1.0), 1: np.float64(1.0)}
```

[5]:
```
# 6. Model configurations with reduced capacity and stronger regularization
configs = [
    {
        'name': 'adam',
        'opt': Adam(learning_rate=1e-5),
        'drop': 0.5,
        'filters': [4, 8],
        'l2_reg': 0.01,
        'dense_units': 16
    },
    {
        'name': 'rmsprop',
        'opt': RMSprop(learning_rate=1e-5),
        'drop': 0.2,
        'filters': [4, 8],
        'l2_reg': 0.02,
        'dense_units': 8
    }
]

results_bin = []
histories = {}
```

[6]:
```
# 7. Spectrogram visualization before training
plt.figure(figsize=(10,4))
plt.imshow(X_train[0,...,0], aspect='auto', cmap='viridis')
plt.title(f"Sample Spectrogram (Class: {y_train[0]})")
plt.colorbar()
plt.show()

for cfg in configs:
    print(f"\nTraining {cfg['name']}...")

    model = Sequential([
```

```
        Conv2D(cfg['filters'][0], (3,3), activation='relu',␣
↪kernel_regularizer=regularizers.l2(cfg['l2_reg']), input_shape=X_train.
↪shape[1:]),
        Dropout(0.3),
        MaxPooling2D((2,2)),
        Conv2D(cfg['filters'][1], (3,3), activation='relu',␣
↪kernel_regularizer=regularizers.l2(cfg['l2_reg'])),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(cfg['dense_units'], activation='relu',␣
↪kernel_regularizer=regularizers.l2(cfg['l2_reg'])),
        Dropout(cfg['drop']),
        Dense(1, activation='sigmoid')
    ])

    model.compile(
        loss='binary_crossentropy',
        optimizer=cfg['opt'],
        metrics=['accuracy', 'Precision', 'Recall', 'AUC']
    )

    print(model.summary())

    es = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)     # es to 5
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
↪min_lr=1e-6)

    history = model.fit(
        datagen.flow(X_train, y_train, batch_size=16, subset='training'), # 16 /
↪ 32
        epochs=60,  # 20 / 100
        validation_data=datagen.flow(X_train, y_train, subset='validation'),
        callbacks=[es, reduce_lr],
        class_weight=cw,
        verbose=1
    )

    # Evaluation with all metrics
    y_prob = model.predict(X_test).ravel()
    y_pred = (y_prob > 0.5).astype(int)

    metrics = {
        'config': cfg['name'],
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred, zero_division=0),
```

```
        'recall': recall_score(y_test, y_pred, zero_division=0),
        'f1_score': f1_score(y_test, y_pred, zero_division=0),
        'roc_auc': roc_auc_score(y_test, y_prob)
    }

    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    metrics['specificity'] = tn / (tn + fp)

    results_bin.append(metrics)
    histories[cfg['name']] = history

    # 8. Prediction distribution plot
    plt.figure()
    plt.hist(y_prob, bins=20)
    plt.title(f"Prediction Distribution - {cfg['name']}")
    plt.xlabel("Predicted Probability")
    plt.ylabel("Count")
    plt.show()
```




```
Training adam…

/opt/homebrew/lib/python3.11/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"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 515, 89, 4)	40
dropout (Dropout)	(None, 515, 89, 4)	0
max_pooling2d (MaxPooling2D)	(None, 257, 44, 4)	0
conv2d_1 (Conv2D)	(None, 255, 42, 8)	296
max_pooling2d_1 (MaxPooling2D)	(None, 127, 21, 8)	0
flatten (Flatten)	(None, 21336)	0
dense (Dense)	(None, 16)	341,392
dropout_1 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1)	17

Total params: 341,745 (1.30 MB)

Trainable params: 341,745 (1.30 MB)

Non-trainable params: 0 (0.00 B)

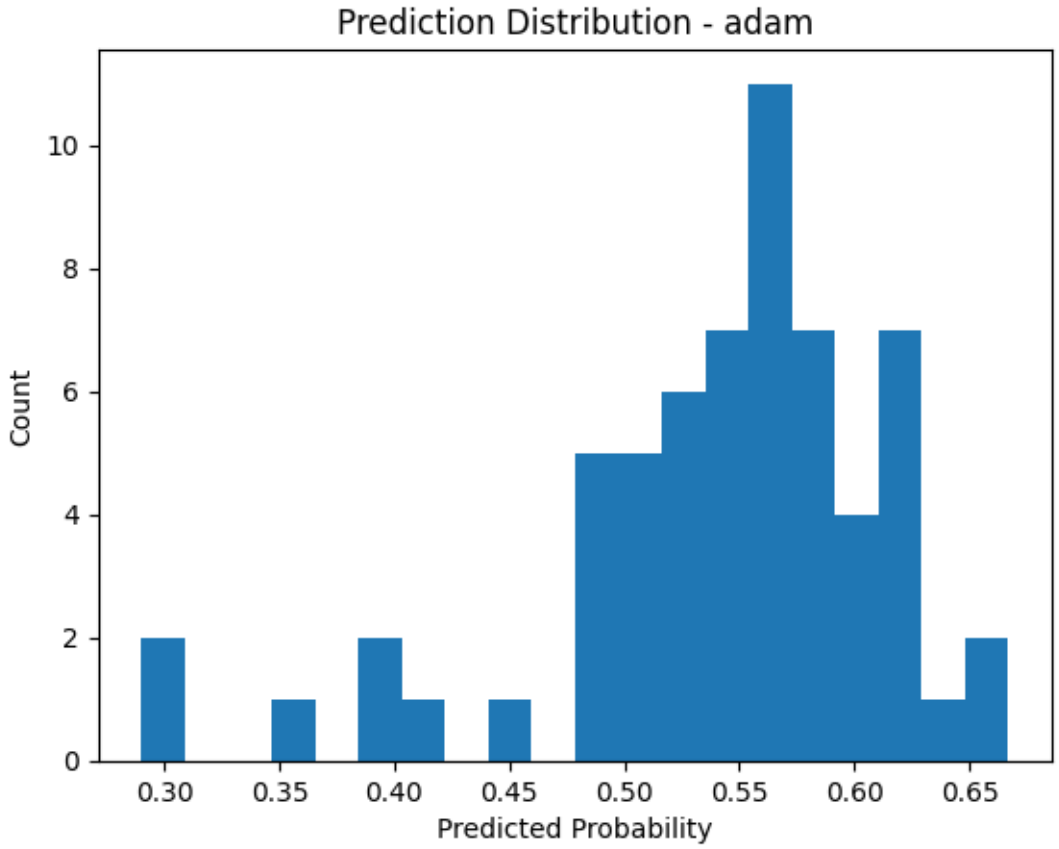
```
None
Epoch 1/60
/opt/homebrew/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
10/10             1s 56ms/step - AUC:
0.6220 - Precision: 0.5979 - Recall: 0.5486 - accuracy: 0.6043 - loss: 1.0636 -
val_AUC: 0.4160 - val_Precision: 0.4583 - val_Recall: 0.6471 - val_accuracy:
0.5000 - val_loss: 1.1041 - learning_rate: 1.0000e-05
Epoch 2/60
10/10             0s 35ms/step - AUC:
```

```
0.5642 - Precision: 0.5455 - Recall: 0.5479 - accuracy: 0.5078 - loss: 1.0748 -
val_AUC: 0.4818 - val_Precision: 0.4815 - val_Recall: 0.7647 - val_accuracy:
0.5263 - val_loss: 1.1006 - learning_rate: 1.0000e-05
Epoch 3/60
10/10            0s 36ms/step - AUC:
0.4379 - Precision: 0.4628 - Recall: 0.6106 - accuracy: 0.4654 - loss: 1.1457 -
val_AUC: 0.4608 - val_Precision: 0.4516 - val_Recall: 0.8235 - val_accuracy:
0.4737 - val_loss: 1.0878 - learning_rate: 1.0000e-05
Epoch 4/60
10/10            0s 35ms/step - AUC:
0.5484 - Precision: 0.5585 - Recall: 0.6787 - accuracy: 0.5605 - loss: 1.0953 -
val_AUC: 0.6218 - val_Precision: 0.5263 - val_Recall: 0.5882 - val_accuracy:
0.5789 - val_loss: 1.0756 - learning_rate: 1.0000e-05
Epoch 5/60
10/10            0s 36ms/step - AUC:
0.7098 - Precision: 0.6550 - Recall: 0.6954 - accuracy: 0.6640 - loss: 1.0350 -
val_AUC: 0.6232 - val_Precision: 0.5385 - val_Recall: 0.8235 - val_accuracy:
0.6053 - val_loss: 1.0649 - learning_rate: 1.0000e-05
Epoch 6/60
10/10            0s 38ms/step - AUC:
0.6935 - Precision: 0.7049 - Recall: 0.6676 - accuracy: 0.6563 - loss: 1.0371 -
val_AUC: 0.7269 - val_Precision: 0.6875 - val_Recall: 0.6471 - val_accuracy:
0.7105 - val_loss: 1.0544 - learning_rate: 1.0000e-05
Epoch 7/60
10/10            0s 35ms/step - AUC:
0.6188 - Precision: 0.5626 - Recall: 0.6553 - accuracy: 0.5678 - loss: 1.0595 -
val_AUC: 0.8627 - val_Precision: 0.6400 - val_Recall: 0.9412 - val_accuracy:
0.7368 - val_loss: 1.0414 - learning_rate: 1.0000e-05
Epoch 8/60
10/10            0s 35ms/step - AUC:
0.6333 - Precision: 0.5384 - Recall: 0.7176 - accuracy: 0.5780 - loss: 1.0660 -
val_AUC: 0.7703 - val_Precision: 0.5926 - val_Recall: 0.9412 - val_accuracy:
0.6842 - val_loss: 1.0497 - learning_rate: 1.0000e-05
Epoch 9/60
10/10            0s 36ms/step - AUC:
0.6847 - Precision: 0.5937 - Recall: 0.8235 - accuracy: 0.6020 - loss: 1.0370 -
val_AUC: 0.7535 - val_Precision: 0.5833 - val_Recall: 0.8235 - val_accuracy:
0.6579 - val_loss: 1.0569 - learning_rate: 1.0000e-05
Epoch 10/60
10/10            0s 40ms/step - AUC:
0.6689 - Precision: 0.6201 - Recall: 0.7718 - accuracy: 0.6255 - loss: 1.0261 -
val_AUC: 0.8417 - val_Precision: 0.6154 - val_Recall: 0.9412 - val_accuracy:
0.7105 - val_loss: 1.0255 - learning_rate: 1.0000e-05
Epoch 11/60
10/10            0s 36ms/step - AUC:
0.6233 - Precision: 0.5571 - Recall: 0.7055 - accuracy: 0.5697 - loss: 1.0618 -
val_AUC: 0.8487 - val_Precision: 0.7000 - val_Recall: 0.8235 - val_accuracy:
0.7632 - val_loss: 1.0110 - learning_rate: 1.0000e-05
```

```
Epoch 12/60
10/10            0s 36ms/step - AUC:
0.6263 - Precision: 0.6040 - Recall: 0.7185 - accuracy: 0.5870 - loss: 1.0382 -
val_AUC: 0.7857 - val_Precision: 0.7500 - val_Recall: 0.7059 - val_accuracy:
0.7632 - val_loss: 1.0337 - learning_rate: 1.0000e-05
Epoch 13/60
10/10            0s 36ms/step - AUC:
0.6178 - Precision: 0.5889 - Recall: 0.7387 - accuracy: 0.5802 - loss: 1.0325 -
val_AUC: 0.8557 - val_Precision: 0.6667 - val_Recall: 0.8235 - val_accuracy:
0.7368 - val_loss: 1.0125 - learning_rate: 1.0000e-05
Epoch 14/60
10/10            0s 36ms/step - AUC:
0.7617 - Precision: 0.6433 - Recall: 0.8464 - accuracy: 0.6871 - loss: 1.0179 -
val_AUC: 0.8627 - val_Precision: 0.6667 - val_Recall: 0.8235 - val_accuracy:
0.7368 - val_loss: 1.0087 - learning_rate: 1.0000e-05
Epoch 15/60
10/10            0s 35ms/step - AUC:
0.7251 - Precision: 0.6300 - Recall: 0.7406 - accuracy: 0.6382 - loss: 1.0061 -
val_AUC: 0.9244 - val_Precision: 0.8750 - val_Recall: 0.8235 - val_accuracy:
0.8684 - val_loss: 1.0004 - learning_rate: 1.0000e-05
Epoch 16/60
10/10            0s 36ms/step - AUC:
0.7348 - Precision: 0.7018 - Recall: 0.6955 - accuracy: 0.6676 - loss: 1.0052 -
val_AUC: 0.8571 - val_Precision: 0.7000 - val_Recall: 0.8235 - val_accuracy:
0.7632 - val_loss: 1.0267 - learning_rate: 1.0000e-05
Epoch 17/60
10/10            0s 35ms/step - AUC:
0.7617 - Precision: 0.6254 - Recall: 0.7770 - accuracy: 0.6767 - loss: 1.0082 -
val_AUC: 0.8319 - val_Precision: 0.8750 - val_Recall: 0.8235 - val_accuracy:
0.8684 - val_loss: 1.0122 - learning_rate: 1.0000e-05
Epoch 18/60
10/10            0s 36ms/step - AUC:
0.7455 - Precision: 0.5942 - Recall: 0.7299 - accuracy: 0.6423 - loss: 0.9940 -
val_AUC: 0.8683 - val_Precision: 0.7500 - val_Recall: 0.8824 - val_accuracy:
0.8158 - val_loss: 0.9900 - learning_rate: 1.0000e-05
Epoch 19/60
10/10            0s 36ms/step - AUC:
0.6756 - Precision: 0.5908 - Recall: 0.6396 - accuracy: 0.5928 - loss: 1.0172 -
val_AUC: 0.6709 - val_Precision: 0.5789 - val_Recall: 0.6471 - val_accuracy:
0.6316 - val_loss: 1.0476 - learning_rate: 1.0000e-05
Epoch 20/60
10/10            0s 36ms/step - AUC:
0.7109 - Precision: 0.6124 - Recall: 0.7419 - accuracy: 0.6237 - loss: 0.9995 -
val_AUC: 0.8866 - val_Precision: 0.8667 - val_Recall: 0.7647 - val_accuracy:
0.8421 - val_loss: 0.9871 - learning_rate: 1.0000e-05
Epoch 21/60
10/10            0s 36ms/step - AUC:
0.8056 - Precision: 0.7070 - Recall: 0.7974 - accuracy: 0.7159 - loss: 0.9651 -
```

```
val_AUC: 0.8109 - val_Precision: 0.6500 - val_Recall: 0.7647 - val_accuracy:
0.7105 - val_loss: 1.0044 - learning_rate: 1.0000e-05
Epoch 22/60
10/10            0s 38ms/step - AUC:
0.8002 - Precision: 0.6804 - Recall: 0.7903 - accuracy: 0.7148 - loss: 0.9634 -
val_AUC: 0.8852 - val_Precision: 0.7000 - val_Recall: 0.8235 - val_accuracy:
0.7632 - val_loss: 0.9527 - learning_rate: 1.0000e-05
Epoch 23/60
10/10            0s 36ms/step - AUC:
0.8039 - Precision: 0.6521 - Recall: 0.7669 - accuracy: 0.6799 - loss: 0.9512 -
val_AUC: 0.8403 - val_Precision: 0.6087 - val_Recall: 0.8235 - val_accuracy:
0.6842 - val_loss: 0.9649 - learning_rate: 1.0000e-05
Epoch 24/60
10/10            0s 36ms/step - AUC:
0.7165 - Precision: 0.6238 - Recall: 0.7000 - accuracy: 0.6304 - loss: 0.9835 -
val_AUC: 0.8683 - val_Precision: 0.6500 - val_Recall: 0.7647 - val_accuracy:
0.7105 - val_loss: 0.9865 - learning_rate: 1.0000e-05
Epoch 25/60
10/10            0s 36ms/step - AUC:
0.7664 - Precision: 0.6658 - Recall: 0.6219 - accuracy: 0.6690 - loss: 0.9735 -
val_AUC: 0.9020 - val_Precision: 0.8125 - val_Recall: 0.7647 - val_accuracy:
0.8158 - val_loss: 0.9577 - learning_rate: 1.0000e-05
Epoch 26/60
10/10            0s 36ms/step - AUC:
0.7374 - Precision: 0.7020 - Recall: 0.6788 - accuracy: 0.6806 - loss: 0.9811 -
val_AUC: 0.9034 - val_Precision: 0.8667 - val_Recall: 0.7647 - val_accuracy:
0.8421 - val_loss: 0.9710 - learning_rate: 5.0000e-06
Epoch 27/60
10/10            0s 36ms/step - AUC:
0.8259 - Precision: 0.8040 - Recall: 0.7526 - accuracy: 0.7587 - loss: 0.9355 -
val_AUC: 0.9412 - val_Precision: 0.7368 - val_Recall: 0.8235 - val_accuracy:
0.7895 - val_loss: 0.9274 - learning_rate: 5.0000e-06
Epoch 28/60
10/10            0s 37ms/step - AUC:
0.7769 - Precision: 0.6480 - Recall: 0.7360 - accuracy: 0.6663 - loss: 0.9375 -
val_AUC: 0.9356 - val_Precision: 0.8824 - val_Recall: 0.8824 - val_accuracy:
0.8947 - val_loss: 0.9402 - learning_rate: 5.0000e-06
Epoch 29/60
10/10            0s 36ms/step - AUC:
0.7720 - Precision: 0.6987 - Recall: 0.7574 - accuracy: 0.6861 - loss: 0.9641 -
val_AUC: 0.8599 - val_Precision: 0.8125 - val_Recall: 0.7647 - val_accuracy:
0.8158 - val_loss: 0.9757 - learning_rate: 5.0000e-06
Epoch 30/60
10/10            0s 36ms/step - AUC:
0.7795 - Precision: 0.6741 - Recall: 0.7069 - accuracy: 0.6615 - loss: 0.9474 -
val_AUC: 0.9062 - val_Precision: 0.9231 - val_Recall: 0.7059 - val_accuracy:
0.8421 - val_loss: 0.9506 - learning_rate: 5.0000e-06
Epoch 31/60
```

```
10/10             0s 36ms/step - AUC:
0.8285 - Precision: 0.7799 - Recall: 0.6873 - accuracy: 0.7303 - loss: 0.9277 -
val_AUC: 0.8459 - val_Precision: 0.7222 - val_Recall: 0.7647 - val_accuracy:
0.7632 - val_loss: 0.9553 - learning_rate: 2.5000e-06
Epoch 32/60
10/10             0s 37ms/step - AUC:
0.7818 - Precision: 0.6646 - Recall: 0.7783 - accuracy: 0.7056 - loss: 0.9677 -
val_AUC: 0.8613 - val_Precision: 0.8750 - val_Recall: 0.8235 - val_accuracy:
0.8684 - val_loss: 0.9404 - learning_rate: 2.5000e-06
2/2             0s 33ms/step
```




```
Training rmsprop…

/opt/homebrew/lib/python3.11/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)
```

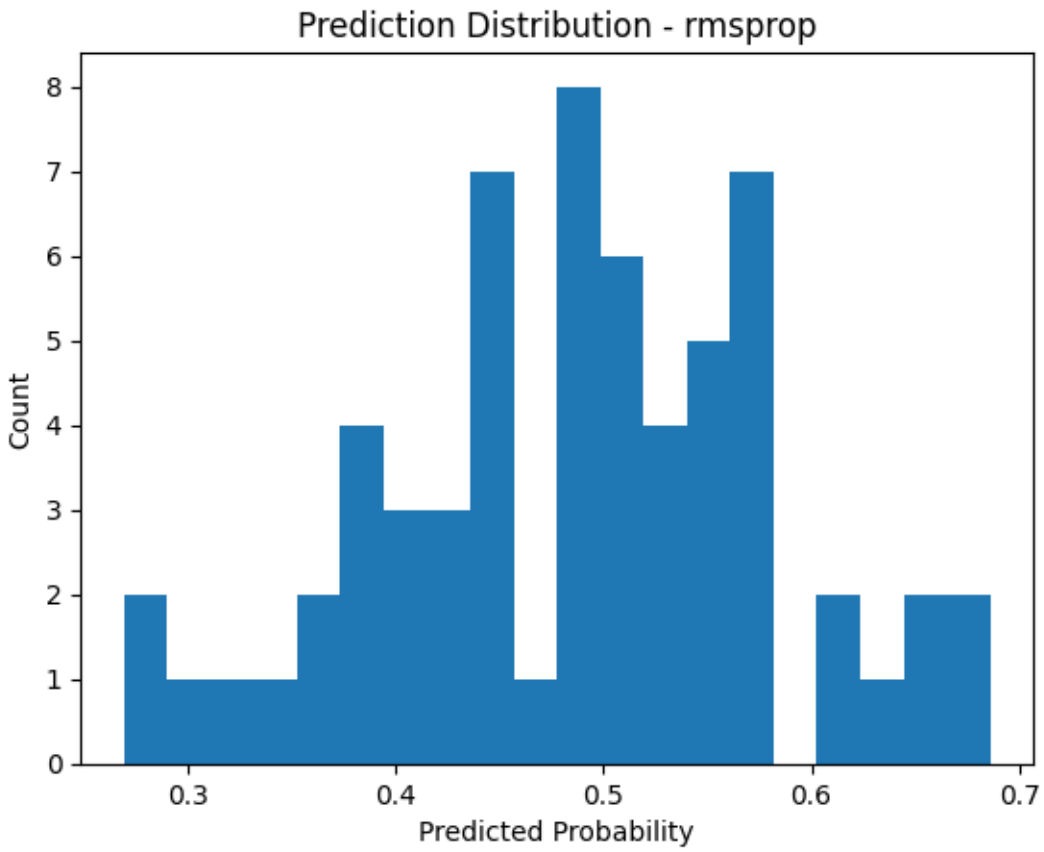
```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 515, 89, 4)	40
dropout_2 (Dropout)	(None, 515, 89, 4)	0
max_pooling2d_2 (MaxPooling2D)	(None, 257, 44, 4)	0
conv2d_3 (Conv2D)	(None, 255, 42, 8)	296
max_pooling2d_3 (MaxPooling2D)	(None, 127, 21, 8)	0
flatten_1 (Flatten)	(None, 21336)	0
dense_2 (Dense)	(None, 8)	170,696
dropout_3 (Dropout)	(None, 8)	0
dense_3 (Dense)	(None, 1)	9

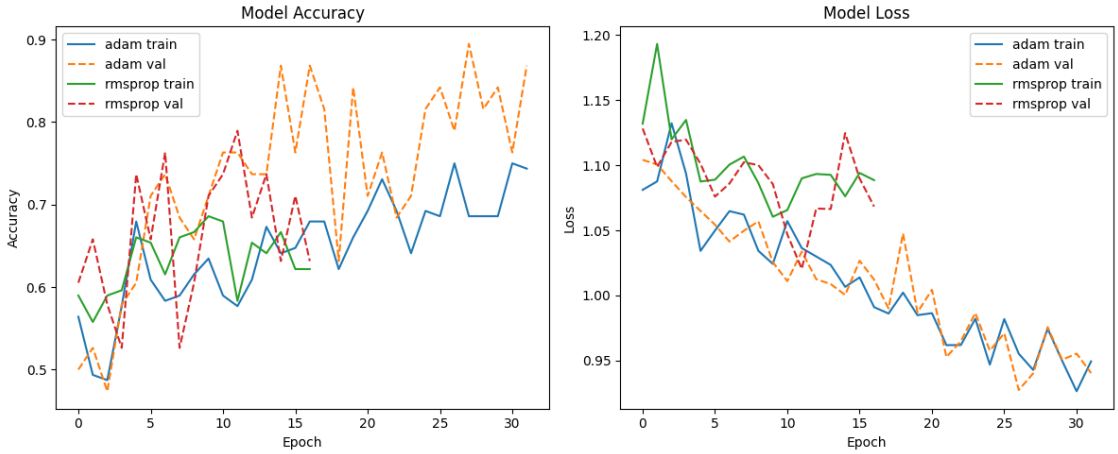
```
 Total params: 171,041 (668.13 KB)

 Trainable params: 171,041 (668.13 KB)

 Non-trainable params: 0 (0.00 B)

None
Epoch 1/60
/opt/homebrew/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
10/10            1s 56ms/step - AUC:
0.5639 - Precision: 0.5369 - Recall: 0.5023 - accuracy: 0.5370 - loss: 1.1913 -
val_AUC: 0.6190 - val_Precision: 0.5556 - val_Recall: 0.5882 - val_accuracy:
0.6053 - val_loss: 1.1282 - learning_rate: 1.0000e-05
Epoch 2/60
```

```
10/10             0s 36ms/step - AUC:
0.4820 - Precision: 0.5365 - Recall: 0.5724 - accuracy: 0.5151 - loss: 1.2371 -
val_AUC: 0.7157 - val_Precision: 0.6667 - val_Recall: 0.4706 - val_accuracy:
0.6579 - val_loss: 1.0987 - learning_rate: 1.0000e-05
Epoch 3/60
10/10             0s 35ms/step - AUC:
0.6083 - Precision: 0.6308 - Recall: 0.5746 - accuracy: 0.5824 - loss: 1.1205 -
val_AUC: 0.6261 - val_Precision: 0.5263 - val_Recall: 0.5882 - val_accuracy:
0.5789 - val_loss: 1.1179 - learning_rate: 1.0000e-05
Epoch 4/60
10/10             0s 35ms/step - AUC:
0.5904 - Precision: 0.5686 - Recall: 0.6222 - accuracy: 0.5538 - loss: 1.1525 -
val_AUC: 0.5994 - val_Precision: 0.4706 - val_Recall: 0.4706 - val_accuracy:
0.5263 - val_loss: 1.1197 - learning_rate: 1.0000e-05
Epoch 5/60
10/10             0s 35ms/step - AUC:
0.6590 - Precision: 0.6363 - Recall: 0.7646 - accuracy: 0.6431 - loss: 1.1192 -
val_AUC: 0.7353 - val_Precision: 0.6842 - val_Recall: 0.7647 - val_accuracy:
0.7368 - val_loss: 1.1012 - learning_rate: 1.0000e-05
Epoch 6/60
10/10             0s 36ms/step - AUC:
0.6596 - Precision: 0.6371 - Recall: 0.6336 - accuracy: 0.6087 - loss: 1.1104 -
val_AUC: 0.7857 - val_Precision: 0.6111 - val_Recall: 0.6471 - val_accuracy:
0.6579 - val_loss: 1.0759 - learning_rate: 5.0000e-06
Epoch 7/60
10/10             0s 35ms/step - AUC:
0.6874 - Precision: 0.6824 - Recall: 0.7055 - accuracy: 0.6518 - loss: 1.0788 -
val_AUC: 0.7815 - val_Precision: 0.7222 - val_Recall: 0.7647 - val_accuracy:
0.7632 - val_loss: 1.0860 - learning_rate: 5.0000e-06
Epoch 8/60
10/10             0s 36ms/step - AUC:
0.7006 - Precision: 0.6536 - Recall: 0.6403 - accuracy: 0.6597 - loss: 1.1062 -
val_AUC: 0.6555 - val_Precision: 0.4615 - val_Recall: 0.3529 - val_accuracy:
0.5263 - val_loss: 1.1023 - learning_rate: 5.0000e-06
Epoch 9/60
10/10             0s 38ms/step - AUC:
0.7223 - Precision: 0.7707 - Recall: 0.6652 - accuracy: 0.7008 - loss: 1.0796 -
val_AUC: 0.6709 - val_Precision: 0.5833 - val_Recall: 0.4118 - val_accuracy:
0.6053 - val_loss: 1.1001 - learning_rate: 5.0000e-06
Epoch 10/60
10/10             0s 36ms/step - AUC:
0.6938 - Precision: 0.7139 - Recall: 0.6510 - accuracy: 0.6811 - loss: 1.0928 -
val_AUC: 0.7101 - val_Precision: 0.6875 - val_Recall: 0.6471 - val_accuracy:
0.7105 - val_loss: 1.0855 - learning_rate: 2.5000e-06
Epoch 11/60
10/10             0s 36ms/step - AUC:
0.7006 - Precision: 0.6540 - Recall: 0.6899 - accuracy: 0.6581 - loss: 1.0789 -
val_AUC: 0.7815 - val_Precision: 0.7333 - val_Recall: 0.6471 - val_accuracy:
```

```
0.7368 - val_loss: 1.0470 - learning_rate: 2.5000e-06
Epoch 12/60
10/10             0s 37ms/step - AUC:
0.7710 - Precision: 0.5817 - Recall: 0.7308 - accuracy: 0.6165 - loss: 1.0534 -
val_AUC: 0.8754 - val_Precision: 0.7647 - val_Recall: 0.7647 - val_accuracy:
0.7895 - val_loss: 1.0204 - learning_rate: 2.5000e-06
Epoch 13/60
10/10             0s 36ms/step - AUC:
0.6781 - Precision: 0.7139 - Recall: 0.5707 - accuracy: 0.6372 - loss: 1.1039 -
val_AUC: 0.7633 - val_Precision: 0.6316 - val_Recall: 0.7059 - val_accuracy:
0.6842 - val_loss: 1.0668 - learning_rate: 2.5000e-06
Epoch 14/60
10/10             0s 35ms/step - AUC:
0.7526 - Precision: 0.7431 - Recall: 0.7101 - accuracy: 0.7211 - loss: 1.0675 -
val_AUC: 0.7647 - val_Precision: 0.6842 - val_Recall: 0.7647 - val_accuracy:
0.7368 - val_loss: 1.0664 - learning_rate: 2.5000e-06
Epoch 15/60
10/10             0s 37ms/step - AUC:
0.6351 - Precision: 0.6420 - Recall: 0.6854 - accuracy: 0.6444 - loss: 1.1028 -
val_AUC: 0.6569 - val_Precision: 0.5652 - val_Recall: 0.7647 - val_accuracy:
0.6316 - val_loss: 1.1247 - learning_rate: 2.5000e-06
Epoch 16/60
10/10             0s 36ms/step - AUC:
0.6585 - Precision: 0.6064 - Recall: 0.6507 - accuracy: 0.6215 - loss: 1.0945 -
val_AUC: 0.7031 - val_Precision: 0.6875 - val_Recall: 0.6471 - val_accuracy:
0.7105 - val_loss: 1.0898 - learning_rate: 1.2500e-06
Epoch 17/60
10/10             0s 37ms/step - AUC:
0.6881 - Precision: 0.5954 - Recall: 0.6704 - accuracy: 0.6088 - loss: 1.0875 -
val_AUC: 0.7241 - val_Precision: 0.6000 - val_Recall: 0.5294 - val_accuracy:
0.6316 - val_loss: 1.0684 - learning_rate: 1.2500e-06
2/2             0s 32ms/step
```

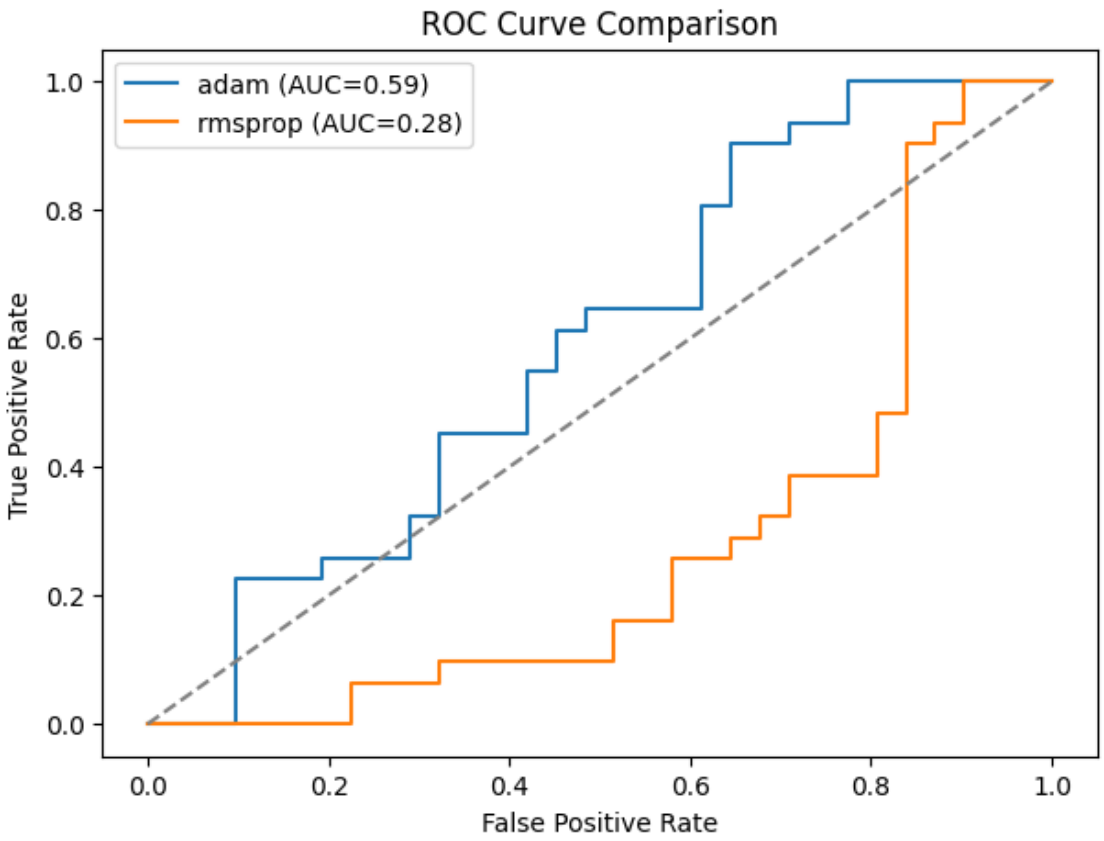


[7]:
```python
# 9. Visualization of results
plt.figure(figsize=(12,5))
# Accuracy
plt.subplot(1,2,1)
for name, h in histories.items():
    plt.plot(h.history['accuracy'], label=f'{name} train')
    plt.plot(h.history['val_accuracy'], '--', label=f'{name} val')
plt.title('Model Accuracy')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.legend()
# Loss
plt.subplot(1,2,2)
for name, h in histories.items():
    plt.plot(h.history['loss'], label=f'{name} train')
    plt.plot(h.history['val_loss'], '--', label=f'{name} val')
plt.title('Model Loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend()
plt.tight_layout(); plt.show()
```



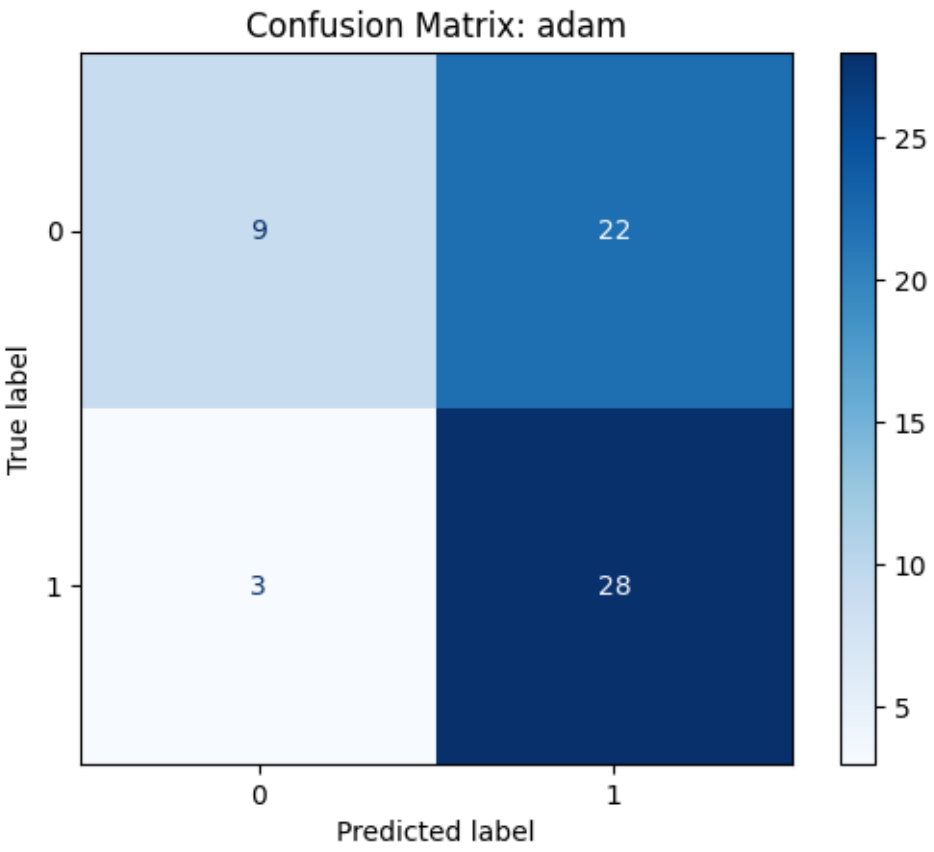
[8]:
```
# 10. ROC Curves
plt.figure(figsize=(7,5))
for res in results_bin:
    name = res['config']
    y_prob = histories[name].model.predict(X_test).ravel()
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC={res['roc_auc']:.2f})")
plt.plot([0,1],[0,1],'--',color='gray')
plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate'); plt.
↪legend(); plt.show()
```

```
2/2             0s 18ms/step
2/2             0s 19ms/step
```

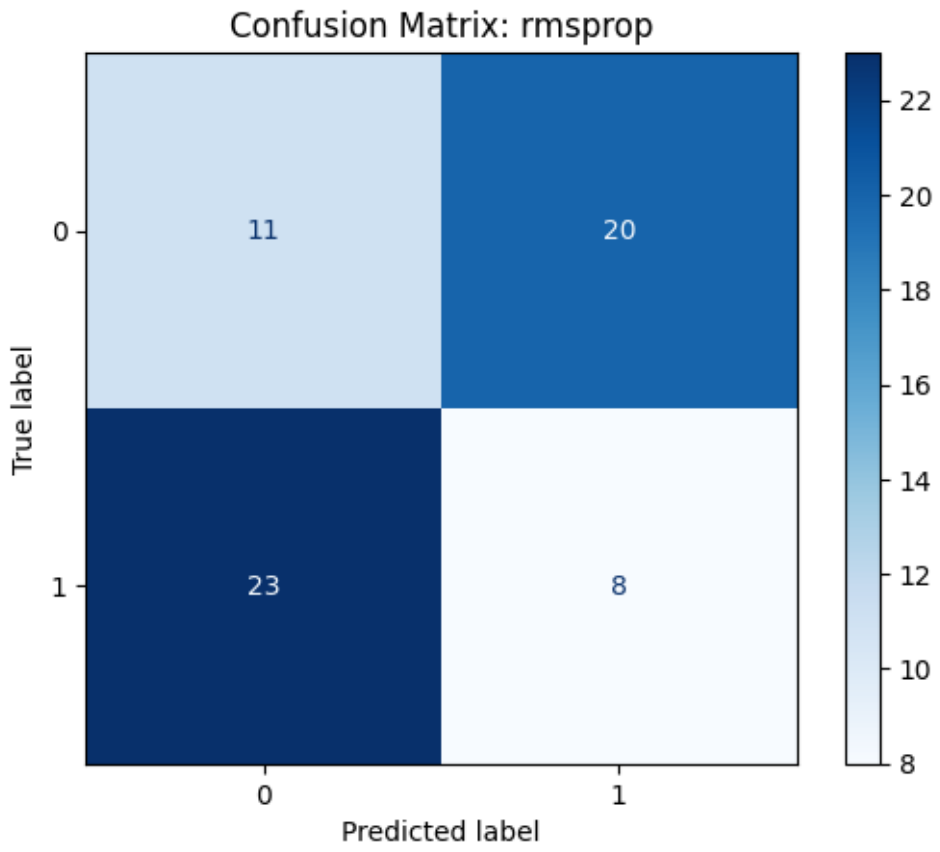


[9]:
```
# 11. Confusion Matrices
for res in results_bin:
    name = res['config']
    y_prob = histories[name].model.predict(X_test).ravel()
    y_pred = (y_prob>0.5).astype(int)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap='Blues')
    plt.title(f"Confusion Matrix: {name}")
    plt.show()
```

```
2/2             0s 19ms/step
```




```
2/2                 0s 18ms/step
```



[10]:
```
# 12. Comparative Results Table
print('\n=== Final Binary Classification Metrics:')
df_bin = pd.DataFrame(results_bin)
display(df_bin.sort_values('roc_auc', ascending=False))
```

```
=== Final Binary Classification Metrics:
```

	config	accuracy	precision	recall	f1_score	roc_auc	specificity
0	adam	0.596774	0.560000	0.903226	0.691358	0.592092	0.290323
1	rmsprop	0.306452	0.285714	0.258065	0.271186	0.280957	0.354839

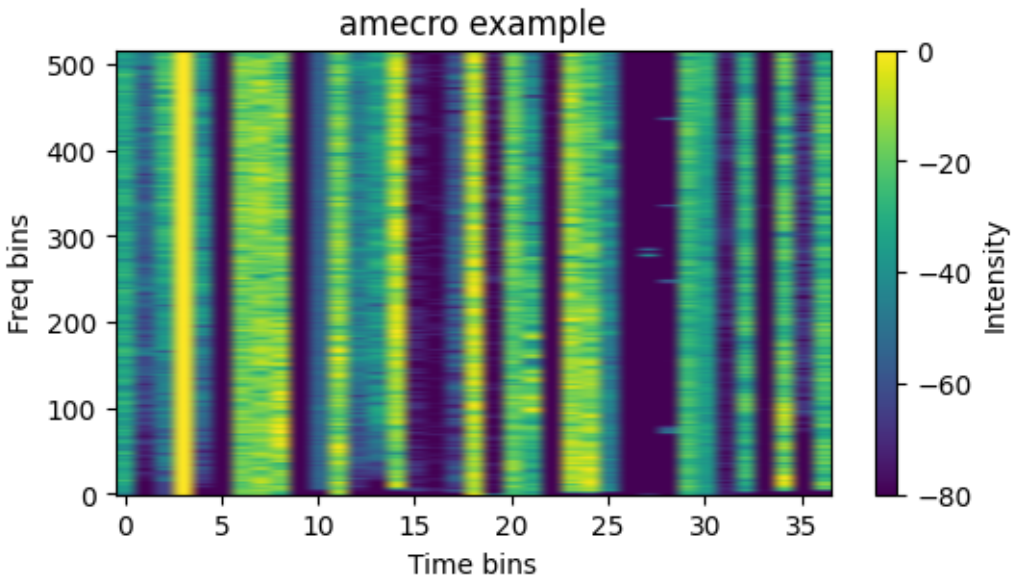
3 2. Multi-class Model

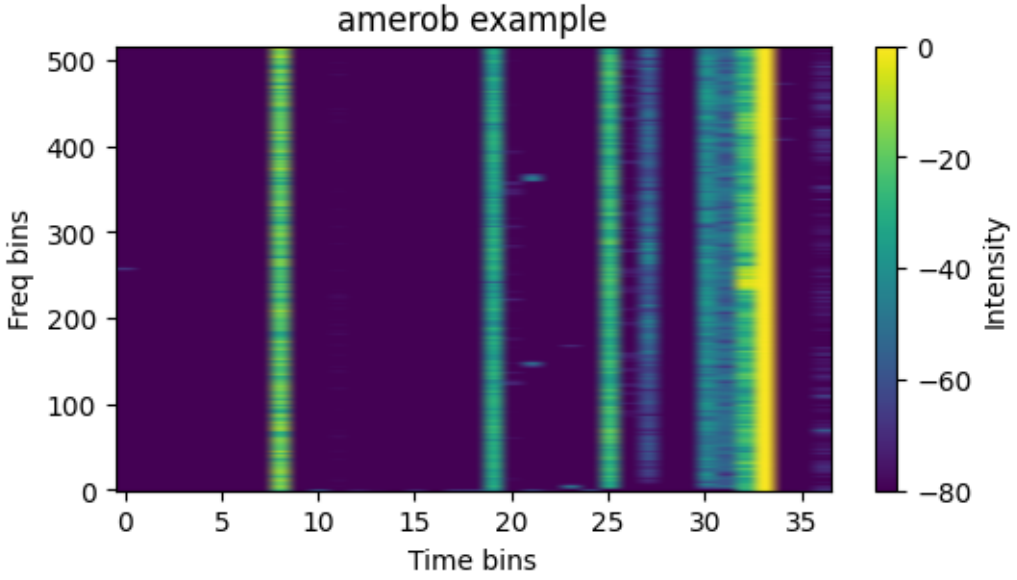
Build a neural network to classify between all 12 bird species.

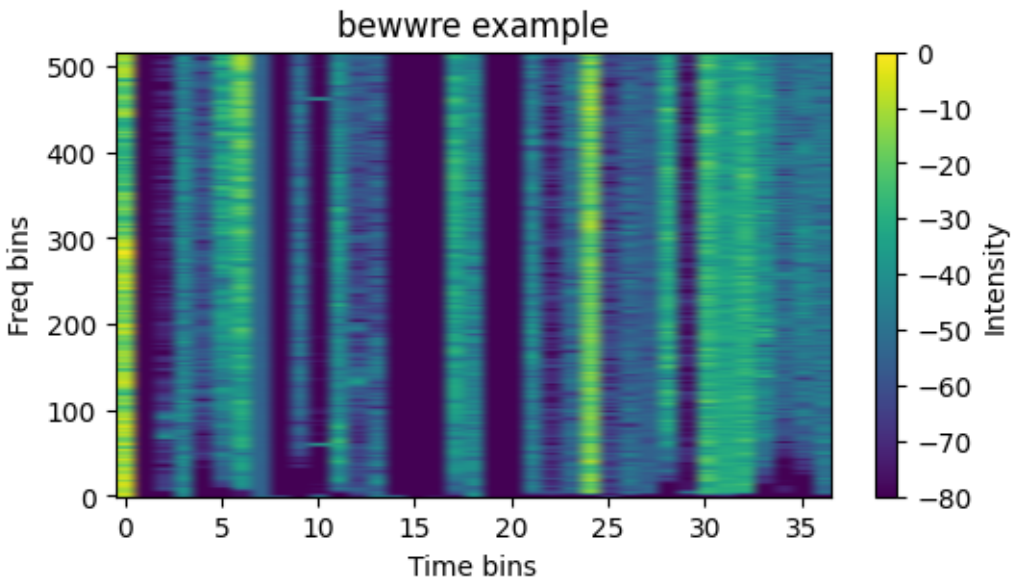
[11]:
```
import time
import numpy as np
import h5py
import matplotlib.pyplot as plt
```

```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle, class_weight
from sklearn.preprocessing import label_binarize
from sklearn.metrics import (accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, roc_curve, confusion_matrix, ConfusionMatrixDisplay)
from sklearn.decomposition import PCA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization)
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, RMSprop
```

[12]:
```
SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)

# 1. Spectrogram for first three species
with h5py.File('bird_spectrograms.hdf5','r') as f:
    species_all = list(f.keys())
    min_t = min(f[sp].shape[2] for sp in species_all)

for sp in species_all[:3]:
    spec = h5py.File('bird_spectrograms.hdf5','r')[sp][0,:,:min_t]
    plt.figure(figsize=(6,3))
    plt.imshow(spec, origin='lower', aspect='auto')
    plt.title(f"{sp} example"); plt.xlabel('Time bins'); plt.ylabel('Freq bins')
    plt.colorbar(label='Intensity'); plt.show()
```


amecro example
Freq bins
Time bins
Intensity
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80


amerob example
Freq bins
Time bins
Intensity
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80



[13]:
```
# 2. Preprocess all species with normalization
def normalize_spectrograms(data):
    return (data - np.mean(data)) / (np.std(data) + 1e-8)

with h5py.File('bird_spectrograms.hdf5','r') as f:
    data, labels = [], []
    for idx, sp in enumerate(species_all):
        arr = normalize_spectrograms(f[sp][:,:,:min_t])
        data.append(arr); labels.append(np.full(len(arr), idx))

X = np.vstack(data)[...,None]
y = np.concatenate(labels)
num_cls = len(species_all)

# 3. Split with data augmentation
X, y = shuffle(X, y, random_state=SEED)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=SEED
)

# Data augmentation
datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    fill_mode='constant',
    cval=0,
    validation_split=0.2
```

```
)

# It converts labels to categorical
y_train_cat = tf.keras.utils.to_categorical(y_train, num_classes=num_cls)
y_test_cat = tf.keras.utils.to_categorical(y_test, num_classes=num_cls)
```

[14]:
```
# 4. Lets try adding Class weights
cls = np.arange(num_cls)
cw_vals = class_weight.compute_class_weight('balanced', classes=cls, y=y_train)
cw_mc = dict(enumerate(cw_vals))
print("Class weights:", cw_mc)
```

```
Class weights: {0: np.float64(0.9935275080906149), 1:
np.float64(1.0032679738562091), 2: np.float64(1.0032679738562091), 3:
np.float64(1.0032679738562091), 4: np.float64(1.0032679738562091), 5:
np.float64(1.0032679738562091), 6: np.float64(1.0032679738562091), 7:
np.float64(1.0032679738562091), 8: np.float64(0.9935275080906149), 9:
np.float64(0.9935275080906149), 10: np.float64(0.9935275080906149), 11:
np.float64(1.0032679738562091)}
```

[15]:
```
# 5. Model Configurations with regularization
configs_mc = [
    {'name':'regularized_simple', 'opt':Adam(learning_rate=1e-4), 'drop':0.3,␣
 ↪'l2_reg':0.001,
     'build': lambda: Sequential([
        Conv2D(32,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.001), input_shape=X_train.shape[1:]),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Dropout(0.2),
        Conv2D(64,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.001)), BatchNormalization(),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪001)),
        Dropout(0.3),
        Dense(num_cls, activation='softmax')
    ])},

    {'name':'deeper_regularized', 'opt':RMSprop(learning_rate=1e-4), 'drop':0.
 ↪5, 'l2_reg':0.002,
     'build': lambda: Sequential([
        Conv2D(32,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.002), input_shape=X_train.shape[1:]),
        BatchNormalization(),
```

```
        Conv2D(32,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.002)),
        MaxPooling2D((2,2)),
        Dropout(0.3),
        Conv2D(64,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.002)), BatchNormalization(),
        Conv2D(64,(3,3), activation='relu', kernel_regularizer=regularizers.
 ↪l2(0.002)),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪002)),
        Dropout(0.5),
        Dense(num_cls, activation='softmax')
    ])}
]

results_mc = []
hist_mc = {}
```

[16]:
```
# 6. Train and Evaluation
for cfg in configs_mc:
    print(f"\nTraining {cfg['name']} configuration...")
    m = cfg['build']()

    m.compile(
        loss='categorical_crossentropy',
        optimizer=cfg['opt'],
        metrics=['accuracy',
                 tf.keras.metrics.Precision(name='precision'),
                 tf.keras.metrics.Recall(name='recall'),
                 tf.keras.metrics.AUC(name='auc')]
    )

    print(m.summary())

    es = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)   # es 2 / 5
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2,␣
 ↪min_lr=1e-6)

    t0 = time.time()
    history = m.fit(
        datagen.flow(X_train, y_train_cat, batch_size=32, subset='training'), #␣
 ↪bs 32
        epochs=120,  # 20 / 100
```

```
        validation_data=datagen.flow(X_train, y_train_cat, subset='validation'),
        callbacks=[es, reduce_lr],
        class_weight=cw_mc,
        verbose=1
    )
    train_time = time.time() - t0
    hist_mc[cfg['name']] = history

    y_prob = m.predict(X_test)
    y_pred = y_prob.argmax(axis=1)

    # All metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='macro', zero_division=0)
    rec = recall_score(y_test, y_pred, average='macro', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)
    auc = roc_auc_score(y_test_cat, y_prob, multi_class='ovr', average='macro')

    results_mc.append({
        'config': cfg['name'],
        'accuracy': acc,
        'precision_macro': prec,
        'recall_macro': rec,
        'f1_macro': f1,
        'roc_auc_macro': auc,
        'train_time_min': round(train_time/60, 2)
    })

    # 8. Confusion matrix
    print(f"\nConfusion matrix for {cfg['name']}:")
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
↪display_labels=species_all)
    disp.plot(cmap='Blues')
    plt.title(f"Confusion Matrix: {cfg['name']}")
    plt.xticks(rotation=45)
    plt.show()

    # 9. Prediction distribution
    plt.figure(figsize=(10,4))
    plt.hist(y_prob.ravel(), bins=50)
    plt.title(f"Prediction Distribution - {cfg['name']}")
    plt.xlabel("Predicted Probability")
    plt.ylabel("Count")
    plt.show()
```

```
Training regularized_simple configuration…
```

```
/opt/homebrew/lib/python3.11/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_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 515, 35, 32)	320
batch_normalization (BatchNormalization)	(None, 515, 35, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 257, 17, 32)	0
dropout_4 (Dropout)	(None, 257, 17, 32)	0
conv2d_5 (Conv2D)	(None, 255, 15, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 255, 15, 64)	256
max_pooling2d_5 (MaxPooling2D)	(None, 127, 7, 64)	0
flatten_2 (Flatten)	(None, 56896)	0
dense_4 (Dense)	(None, 128)	7,282,816
dropout_5 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 12)	1,548

Total params: 7,303,564 (27.86 MB)

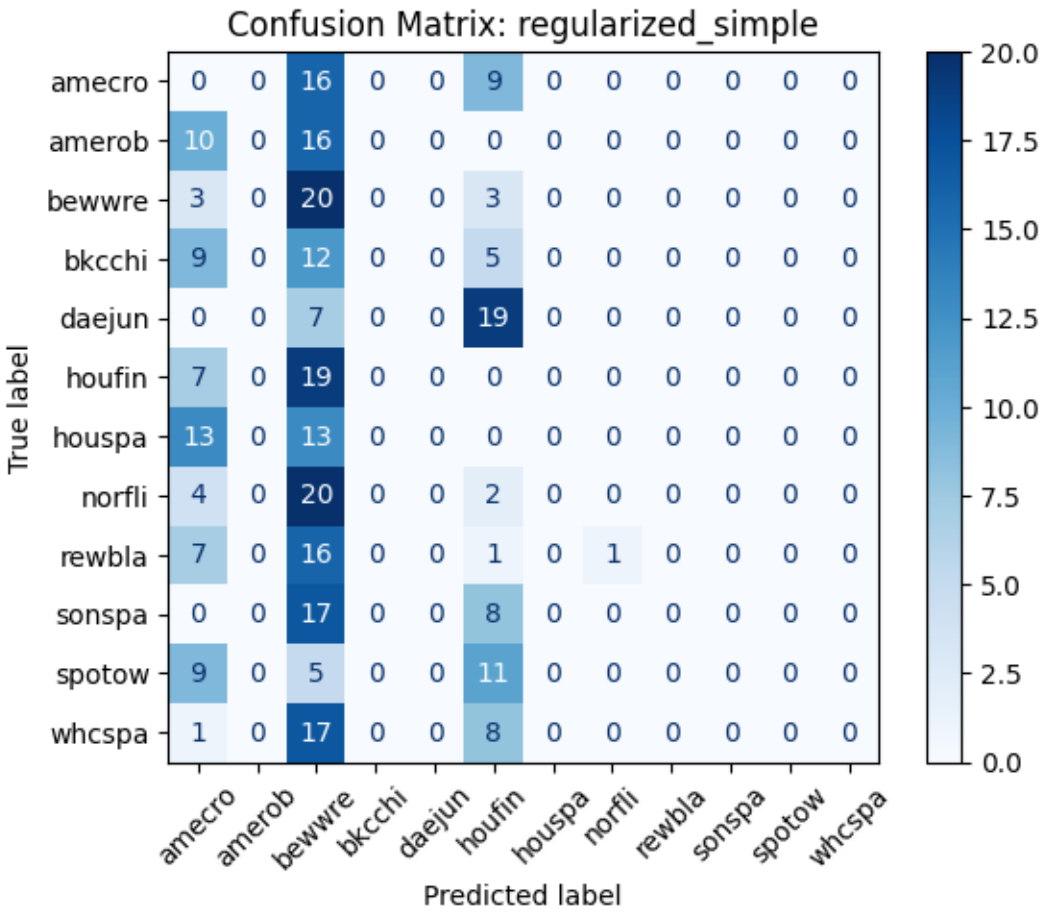
Trainable params: 7,303,372 (27.86 MB)

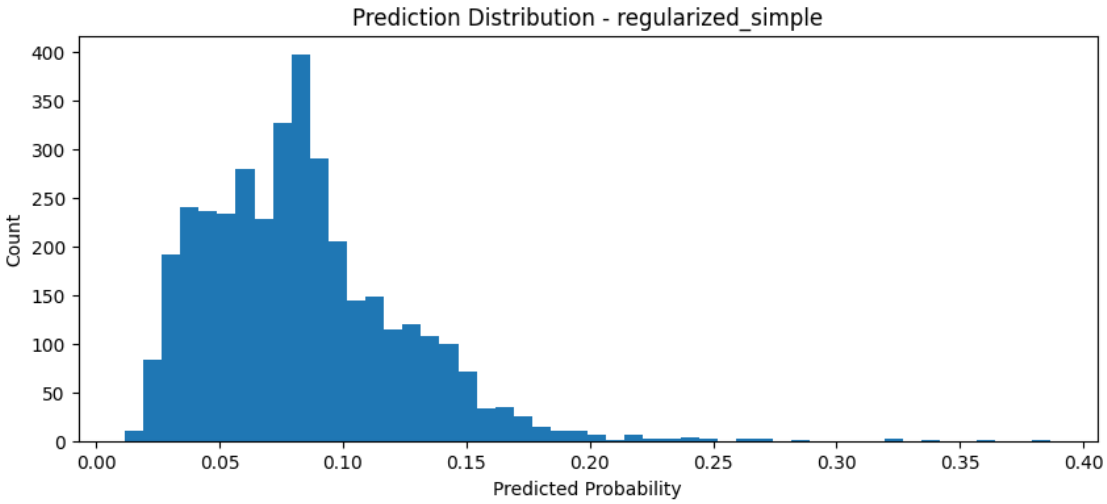
Non-trainable params: 192 (768.00 B)

```
None
Epoch 1/120
```

```
/opt/homebrew/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
31/31            6s 174ms/step -
accuracy: 0.0855 - auc: 0.5073 - loss: 4.8276 - precision: 0.0848 - recall:
0.0342 - val_accuracy: 0.0898 - val_auc: 0.5114 - val_loss: 3.6034 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 2/120
31/31            6s 187ms/step -
accuracy: 0.0869 - auc: 0.5066 - loss: 2.8237 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0980 - val_auc: 0.5258 - val_loss: 5.3472 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 3/120
31/31            6s 181ms/step -
accuracy: 0.0799 - auc: 0.5203 - loss: 2.7807 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0939 - val_auc: 0.5195 - val_loss: 7.0104 -
val_precision: 0.0667 - val_recall: 0.0082 - learning_rate: 1.0000e-04
Epoch 4/120
31/31            6s 183ms/step -
accuracy: 0.0833 - auc: 0.5064 - loss: 2.7778 - precision: 0.3438 - recall:
4.5596e-04 - val_accuracy: 0.0735 - val_auc: 0.5135 - val_loss: 8.3449 -
val_precision: 0.0840 - val_recall: 0.0449 - learning_rate: 5.0000e-05
Epoch 5/120
31/31            6s 197ms/step -
accuracy: 0.0698 - auc: 0.5013 - loss: 2.7844 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0816 - val_auc: 0.5109 - val_loss: 9.7886 -
val_precision: 0.0940 - val_recall: 0.0571 - learning_rate: 5.0000e-05
Epoch 6/120
31/31            6s 182ms/step -
accuracy: 0.0855 - auc: 0.5093 - loss: 2.7791 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0857 - val_auc: 0.5133 - val_loss: 11.0026 -
val_precision: 0.0784 - val_recall: 0.0490 - learning_rate: 2.5000e-05
Epoch 7/120
31/31            6s 191ms/step -
accuracy: 0.0908 - auc: 0.5100 - loss: 2.7746 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0816 - val_auc: 0.5089 - val_loss: 11.8529 -
val_precision: 0.0785 - val_recall: 0.0612 - learning_rate: 2.5000e-05
Epoch 8/120
31/31            6s 192ms/step -
accuracy: 0.0910 - auc: 0.5220 - loss: 2.7848 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0735 - val_auc: 0.5039 - val_loss: 12.2990 -
val_precision: 0.0725 - val_recall: 0.0571 - learning_rate: 1.2500e-05
Epoch 9/120
```

```
31/31              6s 201ms/step -
accuracy: 0.1233 - auc: 0.5228 - loss: 2.7596 - precision: 0.1562 - recall:
1.6568e-04 - val_accuracy: 0.0939 - val_auc: 0.5126 - val_loss: 12.7098 -
val_precision: 0.0882 - val_recall: 0.0735 - learning_rate: 1.2500e-05
Epoch 10/120
31/31              6s 183ms/step -
accuracy: 0.1141 - auc: 0.5196 - loss: 2.7761 - precision: 0.5781 - recall:
0.0024 - val_accuracy: 0.0898 - val_auc: 0.5103 - val_loss: 12.9342 -
val_precision: 0.0849 - val_recall: 0.0735 - learning_rate: 6.2500e-06
Epoch 11/120
31/31              6s 183ms/step -
accuracy: 0.0947 - auc: 0.5180 - loss: 2.7702 - precision: 0.5000 - recall:
0.0010 - val_accuracy: 0.0939 - val_auc: 0.5104 - val_loss: 12.3845 -
val_precision: 0.1106 - val_recall: 0.0939 - learning_rate: 6.2500e-06
10/10              0s 45ms/step

Confusion matrix for regularized_simple:
```






```
Training deeper_regularized configuration…

/opt/homebrew/lib/python3.11/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_3"
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 515, 35, 32)	320
batch_normalization_2 (BatchNormalization)	(None, 515, 35, 32)	128
conv2d_7 (Conv2D)	(None, 513, 33, 32)	9,248
max_pooling2d_6 (MaxPooling2D)	(None, 256, 16, 32)	0
dropout_6 (Dropout)	(None, 256, 16, 32)	0
conv2d_8 (Conv2D)	(None, 254, 14, 64)	18,496
batch_normalization_3 (BatchNormalization)	(None, 254, 14, 64)	256

```
 conv2d_9 (Conv2D)               (None, 252, 12, 64)           36,928

 max_pooling2d_7 (MaxPooling2D)  (None, 126, 6, 64)               0

 flatten_3 (Flatten)             (None, 48384)                    0

 dense_6 (Dense)                 (None, 256)              12,386,560

 dropout_7 (Dropout)             (None, 256)                      0

 dense_7 (Dense)                 (None, 12)                   3,084


 Total params: 12,455,020 (47.51 MB)

 Trainable params: 12,454,828 (47.51 MB)

 Non-trainable params: 192 (768.00 B)

None
Epoch 1/120
/opt/homebrew/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
31/31             15s 469ms/step -
accuracy: 0.0923 - auc: 0.4925 - loss: 9.9017 - precision: 0.0938 - recall:
0.0497 - val_accuracy: 0.0735 - val_auc: 0.4987 - val_loss: 3.7717 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 2/120
31/31             17s 545ms/step -
accuracy: 0.0907 - auc: 0.5008 - loss: 3.7799 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0735 - val_auc: 0.5000 - val_loss: 3.7375 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 3/120
31/31             14s 451ms/step -
accuracy: 0.0805 - auc: 0.5020 - loss: 3.7427 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0694 - val_auc: 0.5000 - val_loss: 3.6825 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 4/120
31/31             16s 512ms/step -
accuracy: 0.0804 - auc: 0.4995 - loss: 3.6710 - precision: 0.0000e+00 - recall:
```

```
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.6157 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 5/120
31/31            14s 445ms/step -
accuracy: 0.0963 - auc: 0.4982 - loss: 3.6200 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.5495 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 6/120
31/31            15s 477ms/step -
accuracy: 0.0810 - auc: 0.5068 - loss: 3.5361 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.4850 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 7/120
31/31            14s 461ms/step -
accuracy: 0.1057 - auc: 0.5079 - loss: 3.4643 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.4255 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 8/120
31/31            15s 476ms/step -
accuracy: 0.0978 - auc: 0.5021 - loss: 3.4121 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.3675 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 9/120
31/31            14s 453ms/step -
accuracy: 0.0995 - auc: 0.5030 - loss: 3.3497 - precision: 0.7812 - recall:
0.0014 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.3120 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 10/120
31/31            16s 513ms/step -
accuracy: 0.0864 - auc: 0.5000 - loss: 3.2975 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.2553 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 11/120
31/31            16s 502ms/step -
accuracy: 0.0731 - auc: 0.4993 - loss: 3.2485 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.2062 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 12/120
31/31            17s 541ms/step -
accuracy: 0.0936 - auc: 0.5006 - loss: 3.1951 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0776 - val_auc: 0.5057 - val_loss: 3.1603 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 13/120
31/31            17s 553ms/step -
accuracy: 0.0770 - auc: 0.4971 - loss: 3.1554 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0694 - val_auc: 0.5020 - val_loss: 3.1217 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 14/120
```

```
31/31            16s 529ms/step -
accuracy: 0.0941 - auc: 0.5025 - loss: 3.1151 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.0847 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 15/120
31/31            16s 520ms/step -
accuracy: 0.0845 - auc: 0.4993 - loss: 3.0758 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.0467 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 16/120
31/31            17s 548ms/step -
accuracy: 0.0721 - auc: 0.4969 - loss: 3.0492 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 3.0117 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 17/120
31/31            16s 519ms/step -
accuracy: 0.0880 - auc: 0.5035 - loss: 3.0065 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5000 - val_loss: 2.9808 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 18/120
31/31            16s 504ms/step -
accuracy: 0.1095 - auc: 0.5026 - loss: 2.9708 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0653 - val_auc: 0.5044 - val_loss: 2.9505 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 19/120
31/31            15s 475ms/step -
accuracy: 0.0900 - auc: 0.5027 - loss: 2.9414 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0816 - val_auc: 0.5057 - val_loss: 2.9227 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 20/120
31/31            14s 460ms/step -
accuracy: 0.0951 - auc: 0.5103 - loss: 2.9086 - precision: 0.6875 - recall:
0.0011 - val_accuracy: 0.0694 - val_auc: 0.5076 - val_loss: 2.8985 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 21/120
31/31            14s 462ms/step -
accuracy: 0.0723 - auc: 0.4973 - loss: 2.8995 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0694 - val_auc: 0.5056 - val_loss: 2.8766 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 22/120
31/31            14s 466ms/step -
accuracy: 0.0845 - auc: 0.5016 - loss: 2.8766 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0735 - val_auc: 0.5210 - val_loss: 2.8471 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 23/120
31/31            14s 464ms/step -
accuracy: 0.1033 - auc: 0.5019 - loss: 2.8508 - precision: 0.0625 - recall:
6.3581e-05 - val_accuracy: 0.0694 - val_auc: 0.5151 - val_loss: 2.8307 -
```

```
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 24/120
31/31             16s 515ms/step -
accuracy: 0.0928 - auc: 0.5025 - loss: 2.8381 - precision: 0.0750 - recall:
4.5375e-04 - val_accuracy: 0.0816 - val_auc: 0.5095 - val_loss: 2.8135 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 25/120
31/31             15s 477ms/step -
accuracy: 0.1107 - auc: 0.5086 - loss: 2.8109 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0816 - val_auc: 0.5015 - val_loss: 2.7979 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 26/120
31/31             16s 502ms/step -
accuracy: 0.1026 - auc: 0.5040 - loss: 2.7917 - precision: 0.7969 - recall:
0.0019 - val_accuracy: 0.0735 - val_auc: 0.5054 - val_loss: 2.7853 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 27/120
31/31             15s 484ms/step -
accuracy: 0.0973 - auc: 0.5045 - loss: 2.7767 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0776 - val_auc: 0.5138 - val_loss: 2.7672 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 28/120
31/31             14s 466ms/step -
accuracy: 0.0998 - auc: 0.5007 - loss: 2.7658 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0816 - val_auc: 0.5172 - val_loss: 2.7478 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 29/120
31/31             16s 510ms/step -
accuracy: 0.0851 - auc: 0.5057 - loss: 2.7444 - precision: 0.7188 - recall:
0.0012 - val_accuracy: 0.0653 - val_auc: 0.5002 - val_loss: 2.7421 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 30/120
31/31             15s 499ms/step -
accuracy: 0.0897 - auc: 0.5075 - loss: 2.7329 - precision: 0.6875 - recall:
0.0020 - val_accuracy: 0.0612 - val_auc: 0.5144 - val_loss: 2.7230 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 31/120
31/31             16s 518ms/step -
accuracy: 0.0834 - auc: 0.5007 - loss: 2.7308 - precision: 0.1385 - recall:
7.7269e-04 - val_accuracy: 0.0694 - val_auc: 0.5118 - val_loss: 2.7157 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 32/120
31/31             16s 521ms/step -
accuracy: 0.0926 - auc: 0.5022 - loss: 2.7135 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0694 - val_auc: 0.5078 - val_loss: 2.7074 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 33/120
31/31             16s 505ms/step -
```

accuracy: 0.0860 - auc: 0.5061 - loss: 2.7005 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0776 - val_auc: 0.5072 - val_loss: 2.6899 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 34/120
31/31 **16s** 506ms/step -
accuracy: 0.0862 - auc: 0.4997 - loss: 2.6878 - precision: 0.3802 - recall:
0.0013 - val_accuracy: 0.0857 - val_auc: 0.5255 - val_loss: 2.6685 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 35/120
31/31 **16s** 518ms/step -
accuracy: 0.1012 - auc: 0.5085 - loss: 2.6785 - precision: 0.4062 - recall:
5.0548e-04 - val_accuracy: 0.0776 - val_auc: 0.5078 - val_loss: 2.6744 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 36/120
31/31 **16s** 512ms/step -
accuracy: 0.0951 - auc: 0.5032 - loss: 2.6742 - precision: 0.0625 - recall:
1.3045e-04 - val_accuracy: 0.0816 - val_auc: 0.5096 - val_loss: 2.6595 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 37/120
31/31 **16s** 506ms/step -
accuracy: 0.0743 - auc: 0.4913 - loss: 2.6713 - precision: 0.3542 - recall:
0.0025 - val_accuracy: 0.0735 - val_auc: 0.5159 - val_loss: 2.6594 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 38/120
31/31 **15s** 488ms/step -
accuracy: 0.0961 - auc: 0.5056 - loss: 2.6608 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_accuracy: 0.0694 - val_auc: 0.5087 - val_loss: 2.6533 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 39/120
31/31 **15s** 474ms/step -
accuracy: 0.0894 - auc: 0.5089 - loss: 2.6437 - precision: 0.8750 - recall:
0.0023 - val_accuracy: 0.0816 - val_auc: 0.5292 - val_loss: 2.6305 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 40/120
31/31 **15s** 485ms/step -
accuracy: 0.0879 - auc: 0.5034 - loss: 2.6432 - precision: 0.3839 - recall:
0.0020 - val_accuracy: 0.0776 - val_auc: 0.4861 - val_loss: 2.6401 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 41/120
31/31 **15s** 487ms/step -
accuracy: 0.0963 - auc: 0.4991 - loss: 2.6384 - precision: 0.5828 - recall:
0.0044 - val_accuracy: 0.0735 - val_auc: 0.5175 - val_loss: 2.6303 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 42/120
31/31 **15s** 486ms/step -
accuracy: 0.0864 - auc: 0.5108 - loss: 2.6318 - precision: 0.8750 - recall:
0.0019 - val_accuracy: 0.0857 - val_auc: 0.5155 - val_loss: 2.6130 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04

```
Epoch 43/120
31/31            16s 512ms/step -
accuracy: 0.0934 - auc: 0.5132 - loss: 2.6141 - precision: 0.6536 - recall:
0.0033 - val_accuracy: 0.0857 - val_auc: 0.5146 - val_loss: 2.6105 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 44/120
31/31            16s 521ms/step -
accuracy: 0.1009 - auc: 0.5192 - loss: 2.6075 - precision: 0.8323 - recall:
0.0047 - val_accuracy: 0.0735 - val_auc: 0.5140 - val_loss: 2.6156 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 45/120
31/31            16s 514ms/step -
accuracy: 0.0930 - auc: 0.5292 - loss: 2.5841 - precision: 0.6676 - recall:
0.0157 - val_accuracy: 0.0735 - val_auc: 0.5212 - val_loss: 2.6032 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 46/120
31/31            17s 549ms/step -
accuracy: 0.1099 - auc: 0.5208 - loss: 2.5971 - precision: 0.8189 - recall:
0.0102 - val_accuracy: 0.0816 - val_auc: 0.5255 - val_loss: 2.5834 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 1.0000e-04
Epoch 47/120
31/31            17s 546ms/step -
accuracy: 0.1205 - auc: 0.5256 - loss: 2.5759 - precision: 0.5597 - recall:
0.0092 - val_accuracy: 0.0939 - val_auc: 0.5359 - val_loss: 2.5759 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 48/120
31/31            18s 565ms/step -
accuracy: 0.1007 - auc: 0.5079 - loss: 2.5978 - precision: 0.2990 - recall:
0.0019 - val_accuracy: 0.0694 - val_auc: 0.5195 - val_loss: 2.5842 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 1.0000e-04
Epoch 49/120
31/31            17s 559ms/step -
accuracy: 0.0954 - auc: 0.5289 - loss: 2.5824 - precision: 0.7991 - recall:
0.0059 - val_accuracy: 0.1020 - val_auc: 0.5409 - val_loss: 2.5645 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 1.0000e-04
Epoch 50/120
31/31            17s 557ms/step -
accuracy: 0.1029 - auc: 0.5147 - loss: 2.5781 - precision: 0.6781 - recall:
0.0075 - val_accuracy: 0.0816 - val_auc: 0.5220 - val_loss: 2.5801 -
val_precision: 1.0000 - val_recall: 0.0041 - learning_rate: 1.0000e-04
Epoch 51/120
31/31            17s 548ms/step -
accuracy: 0.0981 - auc: 0.5250 - loss: 2.5779 - precision: 0.3836 - recall:
0.0060 - val_accuracy: 0.0939 - val_auc: 0.5427 - val_loss: 2.5620 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 1.0000e-04
Epoch 52/120
31/31            18s 574ms/step -
accuracy: 0.1146 - auc: 0.5313 - loss: 2.5601 - precision: 0.4976 - recall:
```

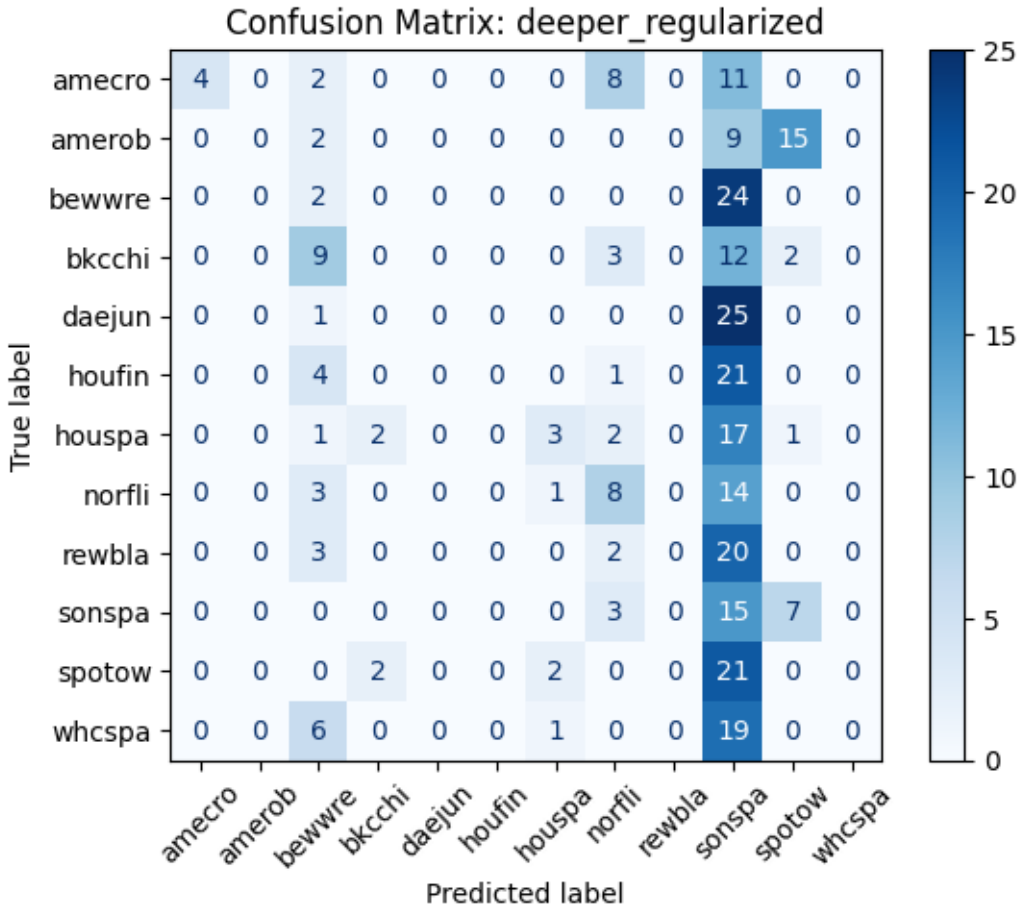
```
0.0069 - val_accuracy: 0.0980 - val_auc: 0.5207 - val_loss: 2.5573 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-04
Epoch 53/120
31/31            17s 567ms/step -
accuracy: 0.1022 - auc: 0.5305 - loss: 2.5580 - precision: 0.6476 - recall:
0.0094 - val_accuracy: 0.1061 - val_auc: 0.5461 - val_loss: 2.5451 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-04
Epoch 54/120
31/31            17s 536ms/step -
accuracy: 0.1320 - auc: 0.5447 - loss: 2.5358 - precision: 0.8365 - recall:
0.0198 - val_accuracy: 0.0857 - val_auc: 0.5261 - val_loss: 2.5649 -
val_precision: 1.0000 - val_recall: 0.0041 - learning_rate: 1.0000e-04
Epoch 55/120
31/31            17s 550ms/step -
accuracy: 0.1077 - auc: 0.5266 - loss: 2.5465 - precision: 0.7105 - recall:
0.0117 - val_accuracy: 0.1102 - val_auc: 0.5527 - val_loss: 2.5726 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-04
Epoch 56/120
31/31            17s 552ms/step -
accuracy: 0.1207 - auc: 0.5321 - loss: 2.5922 - precision: 0.6551 - recall:
0.0109 - val_accuracy: 0.0776 - val_auc: 0.5347 - val_loss: 2.5589 -
val_precision: 0.8000 - val_recall: 0.0163 - learning_rate: 5.0000e-05
Epoch 57/120
31/31            17s 542ms/step -
accuracy: 0.1077 - auc: 0.5223 - loss: 2.5771 - precision: 0.3213 - recall:
0.0049 - val_accuracy: 0.1102 - val_auc: 0.5583 - val_loss: 2.5276 -
val_precision: 0.8571 - val_recall: 0.0245 - learning_rate: 5.0000e-05
Epoch 58/120
31/31            16s 512ms/step -
accuracy: 0.1214 - auc: 0.5389 - loss: 2.5455 - precision: 0.5709 - recall:
0.0068 - val_accuracy: 0.1224 - val_auc: 0.5379 - val_loss: 2.5491 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 5.0000e-05
Epoch 59/120
31/31            18s 568ms/step -
accuracy: 0.1094 - auc: 0.5328 - loss: 2.5543 - precision: 0.6746 - recall:
0.0107 - val_accuracy: 0.1265 - val_auc: 0.5451 - val_loss: 2.5143 -
val_precision: 1.0000 - val_recall: 0.0245 - learning_rate: 5.0000e-05
Epoch 60/120
31/31            18s 574ms/step -
accuracy: 0.1354 - auc: 0.5488 - loss: 2.5409 - precision: 0.8531 - recall:
0.0096 - val_accuracy: 0.0816 - val_auc: 0.5595 - val_loss: 2.5466 -
val_precision: 0.5714 - val_recall: 0.0163 - learning_rate: 5.0000e-05
Epoch 61/120
31/31            18s 573ms/step -
accuracy: 0.1126 - auc: 0.5472 - loss: 2.5240 - precision: 0.7287 - recall:
0.0212 - val_accuracy: 0.1184 - val_auc: 0.5952 - val_loss: 2.4993 -
val_precision: 0.7143 - val_recall: 0.0204 - learning_rate: 5.0000e-05
Epoch 62/120
```

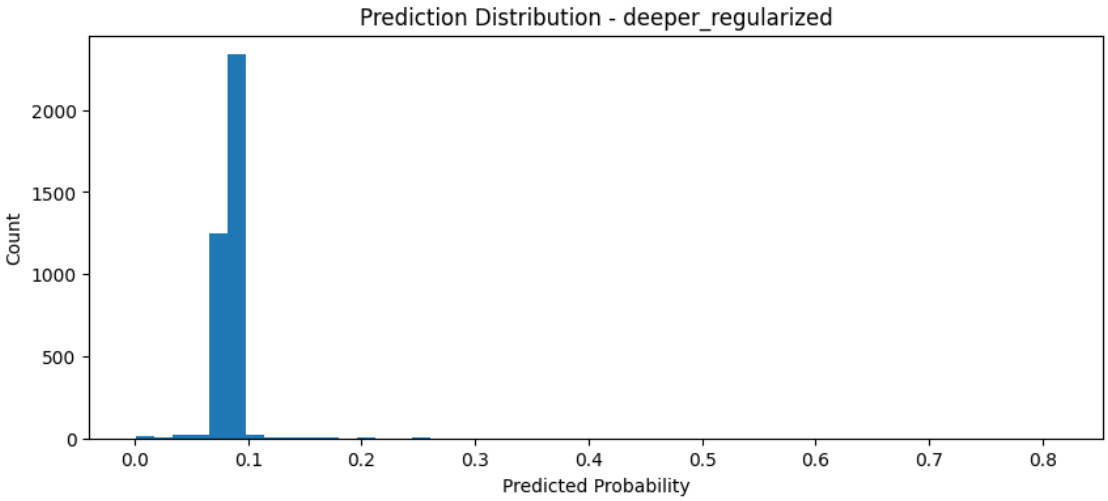
```
31/31            19s 614ms/step -
accuracy: 0.1142 - auc: 0.5451 - loss: 2.5216 - precision: 0.7555 - recall:
0.0173 - val_accuracy: 0.0980 - val_auc: 0.5745 - val_loss: 2.5197 -
val_precision: 0.6250 - val_recall: 0.0204 - learning_rate: 5.0000e-05
Epoch 63/120
31/31            17s 562ms/step -
accuracy: 0.1145 - auc: 0.5356 - loss: 2.5652 - precision: 0.2462 - recall:
0.0043 - val_accuracy: 0.1224 - val_auc: 0.5754 - val_loss: 2.5156 -
val_precision: 0.6667 - val_recall: 0.0082 - learning_rate: 5.0000e-05
Epoch 64/120
31/31            18s 586ms/step -
accuracy: 0.1094 - auc: 0.5495 - loss: 2.5305 - precision: 0.6490 - recall:
0.0156 - val_accuracy: 0.1306 - val_auc: 0.5335 - val_loss: 2.5542 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 2.5000e-05
Epoch 65/120
31/31            17s 562ms/step -
accuracy: 0.1326 - auc: 0.5591 - loss: 2.5036 - precision: 0.7072 - recall:
0.0139 - val_accuracy: 0.1224 - val_auc: 0.5524 - val_loss: 2.5276 -
val_precision: 0.6667 - val_recall: 0.0082 - learning_rate: 2.5000e-05
Epoch 66/120
31/31            17s 547ms/step -
accuracy: 0.1353 - auc: 0.5542 - loss: 2.5221 - precision: 0.5957 - recall:
0.0108 - val_accuracy: 0.1469 - val_auc: 0.5812 - val_loss: 2.5194 -
val_precision: 0.6000 - val_recall: 0.0122 - learning_rate: 1.2500e-05
Epoch 67/120
31/31            17s 555ms/step -
accuracy: 0.1110 - auc: 0.5501 - loss: 2.5199 - precision: 0.6842 - recall:
0.0146 - val_accuracy: 0.1143 - val_auc: 0.5476 - val_loss: 2.5311 -
val_precision: 1.0000 - val_recall: 0.0163 - learning_rate: 1.2500e-05
Epoch 68/120
31/31            17s 555ms/step -
accuracy: 0.1318 - auc: 0.5772 - loss: 2.5046 - precision: 0.6417 - recall:
0.0145 - val_accuracy: 0.1184 - val_auc: 0.5710 - val_loss: 2.4820 -
val_precision: 1.0000 - val_recall: 0.0367 - learning_rate: 6.2500e-06
Epoch 69/120
31/31            17s 539ms/step -
accuracy: 0.1032 - auc: 0.5490 - loss: 2.5199 - precision: 0.6914 - recall:
0.0127 - val_accuracy: 0.1184 - val_auc: 0.5704 - val_loss: 2.5229 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 6.2500e-06
Epoch 70/120
31/31            17s 543ms/step -
accuracy: 0.1286 - auc: 0.5667 - loss: 2.5080 - precision: 0.6996 - recall:
0.0130 - val_accuracy: 0.1551 - val_auc: 0.5647 - val_loss: 2.5283 -
val_precision: 0.5000 - val_recall: 0.0082 - learning_rate: 6.2500e-06
Epoch 71/120
31/31            17s 560ms/step -
accuracy: 0.1322 - auc: 0.5506 - loss: 2.5172 - precision: 0.5174 - recall:
0.0097 - val_accuracy: 0.1551 - val_auc: 0.5804 - val_loss: 2.4931 -
```

```
val_precision: 0.7143 - val_recall: 0.0204 - learning_rate: 3.1250e-06
Epoch 72/120
31/31            17s 556ms/step -
accuracy: 0.1319 - auc: 0.5529 - loss: 2.5301 - precision: 0.5361 - recall:
0.0061 - val_accuracy: 0.0980 - val_auc: 0.5669 - val_loss: 2.5244 -
val_precision: 0.6667 - val_recall: 0.0163 - learning_rate: 3.1250e-06
Epoch 73/120
31/31            17s 548ms/step -
accuracy: 0.1202 - auc: 0.5594 - loss: 2.5112 - precision: 0.5975 - recall:
0.0120 - val_accuracy: 0.1061 - val_auc: 0.6036 - val_loss: 2.4806 -
val_precision: 1.0000 - val_recall: 0.0245 - learning_rate: 1.5625e-06
Epoch 74/120
31/31            19s 601ms/step -
accuracy: 0.1393 - auc: 0.5612 - loss: 2.5199 - precision: 0.5421 - recall:
0.0126 - val_accuracy: 0.1020 - val_auc: 0.5897 - val_loss: 2.4828 -
val_precision: 1.0000 - val_recall: 0.0204 - learning_rate: 1.5625e-06
Epoch 75/120
31/31            17s 538ms/step -
accuracy: 0.1225 - auc: 0.5658 - loss: 2.5097 - precision: 0.7842 - recall:
0.0206 - val_accuracy: 0.1061 - val_auc: 0.5651 - val_loss: 2.5188 -
val_precision: 0.7500 - val_recall: 0.0122 - learning_rate: 1.5625e-06
Epoch 76/120
31/31            17s 560ms/step -
accuracy: 0.1398 - auc: 0.5699 - loss: 2.5021 - precision: 0.8088 - recall:
0.0177 - val_accuracy: 0.1224 - val_auc: 0.5608 - val_loss: 2.4966 -
val_precision: 0.8000 - val_recall: 0.0163 - learning_rate: 1.0000e-06
Epoch 77/120
31/31            17s 563ms/step -
accuracy: 0.1359 - auc: 0.5818 - loss: 2.4995 - precision: 0.6475 - recall:
0.0167 - val_accuracy: 0.1429 - val_auc: 0.6131 - val_loss: 2.5065 -
val_precision: 0.5556 - val_recall: 0.0204 - learning_rate: 1.0000e-06
Epoch 78/120
31/31            17s 561ms/step -
accuracy: 0.1259 - auc: 0.5772 - loss: 2.4833 - precision: 0.7526 - recall:
0.0253 - val_accuracy: 0.1224 - val_auc: 0.5869 - val_loss: 2.4828 -
val_precision: 0.7778 - val_recall: 0.0286 - learning_rate: 1.0000e-06
Epoch 79/120
31/31            17s 546ms/step -
accuracy: 0.1154 - auc: 0.5497 - loss: 2.5303 - precision: 0.6516 - recall:
0.0144 - val_accuracy: 0.1469 - val_auc: 0.5696 - val_loss: 2.4712 -
val_precision: 0.9091 - val_recall: 0.0408 - learning_rate: 1.0000e-06
Epoch 80/120
31/31            17s 559ms/step -
accuracy: 0.1301 - auc: 0.5731 - loss: 2.5001 - precision: 0.6105 - recall:
0.0196 - val_accuracy: 0.1265 - val_auc: 0.5943 - val_loss: 2.5169 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 1.0000e-06
Epoch 81/120
31/31            17s 554ms/step -
```

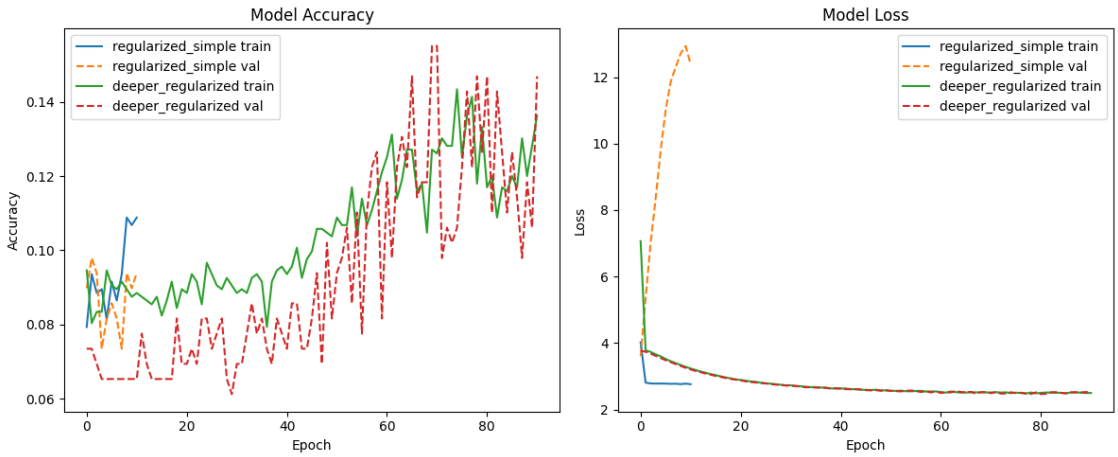
```
accuracy: 0.1099 - auc: 0.5598 - loss: 2.5211 - precision: 0.6980 - recall:
0.0137 - val_accuracy: 0.1469 - val_auc: 0.5940 - val_loss: 2.4635 -
val_precision: 1.0000 - val_recall: 0.0327 - learning_rate: 1.0000e-06
Epoch 82/120
31/31            17s 562ms/step -
accuracy: 0.1245 - auc: 0.5741 - loss: 2.4969 - precision: 0.4811 - recall:
0.0140 - val_accuracy: 0.1102 - val_auc: 0.5891 - val_loss: 2.4769 -
val_precision: 1.0000 - val_recall: 0.0286 - learning_rate: 1.0000e-06
Epoch 83/120
31/31            17s 540ms/step -
accuracy: 0.0918 - auc: 0.5434 - loss: 2.5353 - precision: 0.5543 - recall:
0.0096 - val_accuracy: 0.1429 - val_auc: 0.5812 - val_loss: 2.5193 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-06
Epoch 84/120
31/31            16s 510ms/step -
accuracy: 0.1179 - auc: 0.5607 - loss: 2.5114 - precision: 0.7590 - recall:
0.0176 - val_accuracy: 0.1265 - val_auc: 0.5686 - val_loss: 2.5233 -
val_precision: 0.7500 - val_recall: 0.0122 - learning_rate: 1.0000e-06
Epoch 85/120
31/31            16s 520ms/step -
accuracy: 0.1134 - auc: 0.5795 - loss: 2.5050 - precision: 0.6356 - recall:
0.0147 - val_accuracy: 0.1102 - val_auc: 0.5660 - val_loss: 2.5013 -
val_precision: 1.0000 - val_recall: 0.0204 - learning_rate: 1.0000e-06
Epoch 86/120
31/31            16s 529ms/step -
accuracy: 0.1169 - auc: 0.5666 - loss: 2.4993 - precision: 0.7101 - recall:
0.0171 - val_accuracy: 0.1265 - val_auc: 0.5780 - val_loss: 2.4848 -
val_precision: 1.0000 - val_recall: 0.0245 - learning_rate: 1.0000e-06
Epoch 87/120
31/31            16s 531ms/step -
accuracy: 0.1232 - auc: 0.5765 - loss: 2.4919 - precision: 0.8118 - recall:
0.0193 - val_accuracy: 0.1143 - val_auc: 0.5322 - val_loss: 2.5166 -
val_precision: 1.0000 - val_recall: 0.0204 - learning_rate: 1.0000e-06
Epoch 88/120
31/31            16s 523ms/step -
accuracy: 0.1351 - auc: 0.5773 - loss: 2.5136 - precision: 0.8121 - recall:
0.0108 - val_accuracy: 0.0980 - val_auc: 0.5920 - val_loss: 2.5199 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-06
Epoch 89/120
31/31            16s 506ms/step -
accuracy: 0.1295 - auc: 0.5611 - loss: 2.5029 - precision: 0.8357 - recall:
0.0174 - val_accuracy: 0.1184 - val_auc: 0.5728 - val_loss: 2.5163 -
val_precision: 1.0000 - val_recall: 0.0122 - learning_rate: 1.0000e-06
Epoch 90/120
31/31            16s 523ms/step -
accuracy: 0.1238 - auc: 0.5690 - loss: 2.4903 - precision: 0.7469 - recall:
0.0197 - val_accuracy: 0.1061 - val_auc: 0.5943 - val_loss: 2.5275 -
val_precision: 1.0000 - val_recall: 0.0082 - learning_rate: 1.0000e-06
```

```
Epoch 91/120
31/31              20s 656ms/step -
accuracy: 0.1436 - auc: 0.5806 - loss: 2.5009 - precision: 0.7296 - recall:
0.0197 - val_accuracy: 0.1469 - val_auc: 0.5464 - val_loss: 2.5062 -
val_precision: 1.0000 - val_recall: 0.0163 - learning_rate: 1.0000e-06
10/10              1s 114ms/step

Confusion matrix for deeper_regularized:
```



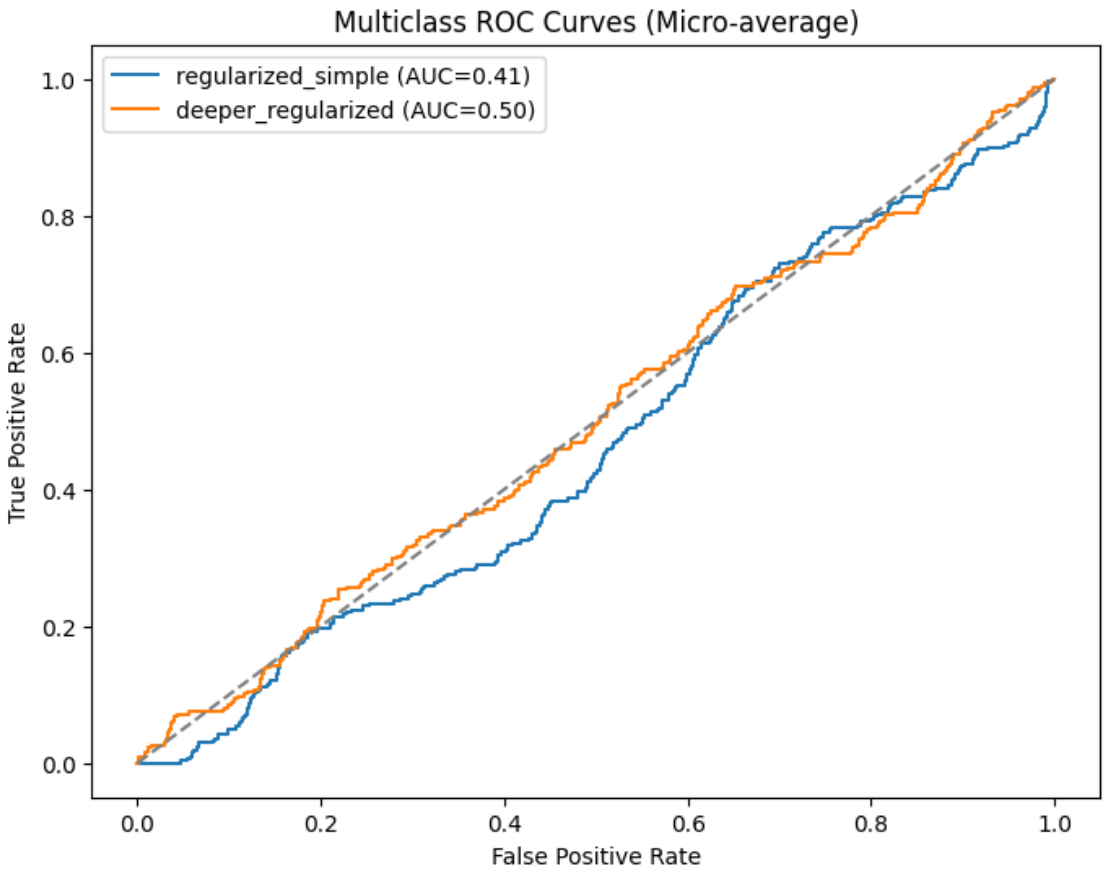


[17]:
```
# 10. Lets plot - Loss and Accuracy vs Epochs
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
for name, h in hist_mc.items():
    plt.plot(h.history['accuracy'], label=f'{name} train')
    plt.plot(h.history['val_accuracy'], '--', label=f'{name} val')
plt.title('Model Accuracy'); plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.
 ↪legend()

plt.subplot(1,2,2)
for name, h in hist_mc.items():
    plt.plot(h.history['loss'], label=f'{name} train')
    plt.plot(h.history['val_loss'], '--', label=f'{name} val')
plt.title('Model Loss'); plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend()
plt.tight_layout(); plt.show()
```



[18]:
```
# 11. ROC curves
plt.figure(figsize=(8,6))
for res in results_mc:
    name = res['config']
    y_prob = hist_mc[name].model.predict(X_test)
    fpr, tpr, _ = roc_curve(y_test_cat.ravel(), y_prob.ravel())
    plt.plot(fpr, tpr, label=f"{name} (AUC={res['roc_auc_macro']:.2f})")
plt.plot([0,1],[0,1],'--',color='gray')
plt.title('Multiclass ROC Curves (Micro-average)')
plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate')
plt.legend(); plt.show()
```

```
10/10             0s 34ms/step
10/10             1s 100ms/step
```



[19]:
```
# 12. Comparative Results table
print('\n=== Final Multiclass Classification Metrics:')
```

```
df_mc = pd.DataFrame(results_mc)
display(df_mc.sort_values('roc_auc_macro', ascending=False))
```

```
=== Final Multiclass Classification Metrics:

               config  accuracy  precision_macro  recall_macro  f1_macro  \
1  deeper_regularized  0.103896         0.154799      0.105000  0.079677
0  regularized_simple  0.064935         0.009363      0.064103  0.016340

   roc_auc_macro  train_time_min
1       0.504442           24.73
0       0.407106            1.08
```

4 3. External Test Data

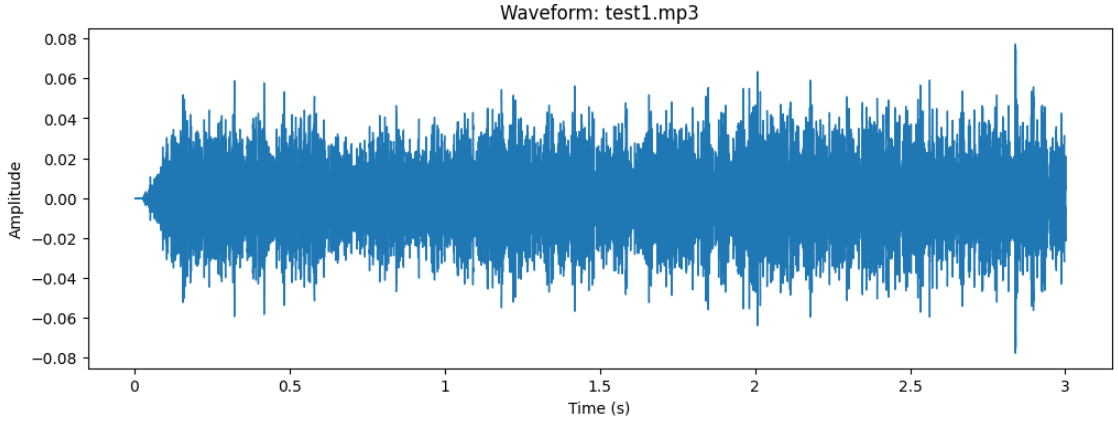
Each of 3 test clips provided are of raw sound data (mp3), some of which contain more than one bird call. Convert them using the methodology described below, and use your 12-species network to predict which birds are calling– clearly state your prediction of each of the three clips in a table in your results section. Which clips do you think contain more than one bird and why? Make sure to justify your reasoning with a plot or data.

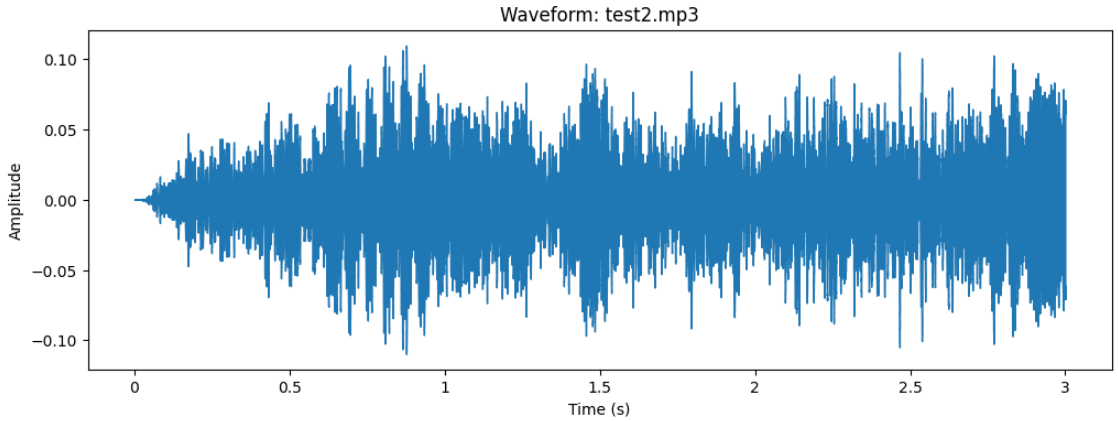
[24]:
```
import time
import glob
import numpy as np
import pandas as pd
import librosa
import h5py
import librosa.display
import matplotlib.pyplot as plt
```

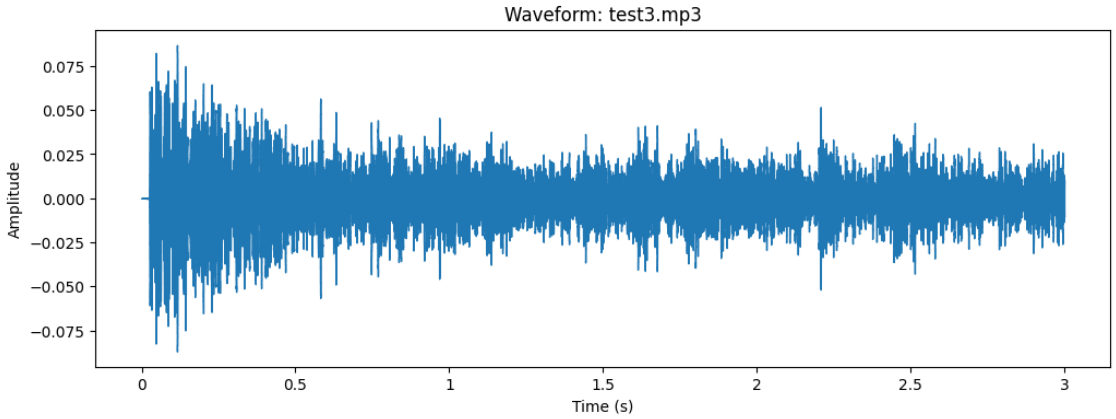
[25]:
```
# 1. Considering the model and parameters of window frame for audio file
with h5py.File('bird_spectrograms.hdf5','r') as f:
    species_all = list(f.keys())
    min_t = min(f[sp].shape[2] for sp in species_all)
    freq_bins = f[species_all[0]].shape[1]


sr = 22050; dur=3.0; win_s=2.0; hop_s=1.0
win_n=int(sr*win_s); hop_n=int(sr*hop_s)
rows = []
```

[26]:
```
# 2. Waveform visualization for each of the test audio
def show_waveform(path):
    y, sr = librosa.load(path, duration=3.0)
    plt.figure(figsize=(12, 4))
    librosa.display.waveshow(y, sr=sr)
    plt.title(f"Waveform: {path.split('/')[-1]}")
```

```
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.show()

for path in sorted(glob.glob('test_birds/*.mp3')):
    show_waveform(path)
```

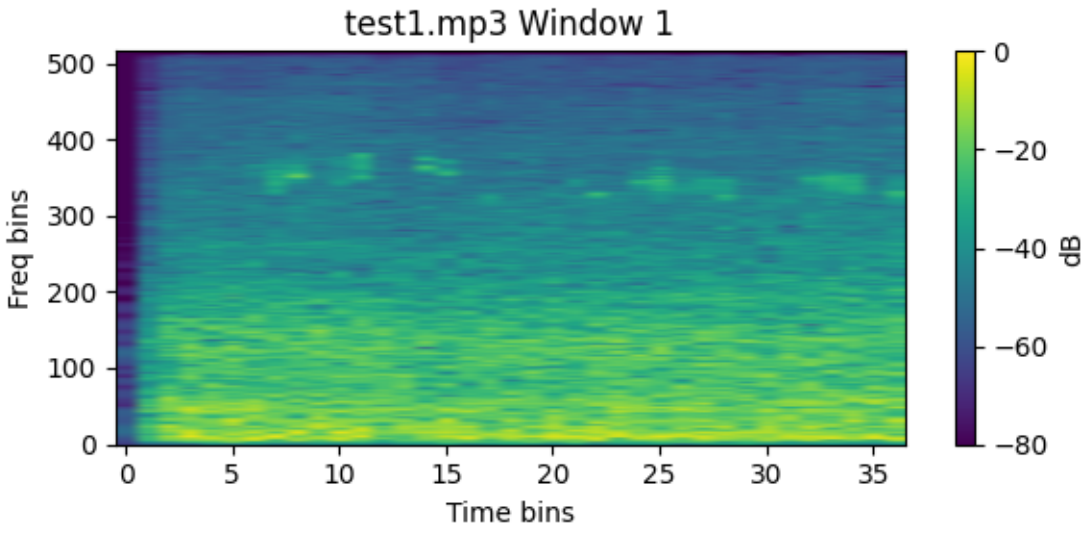


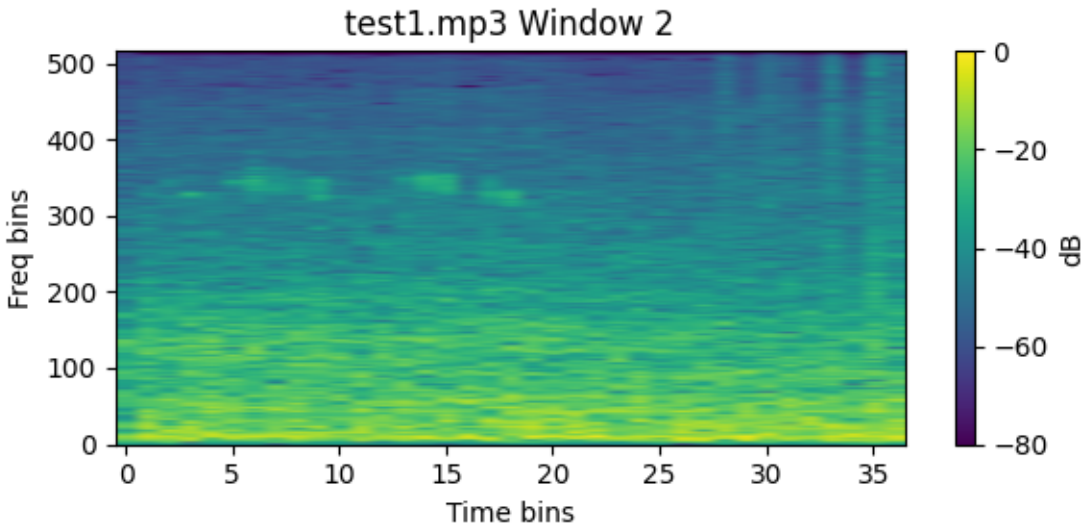


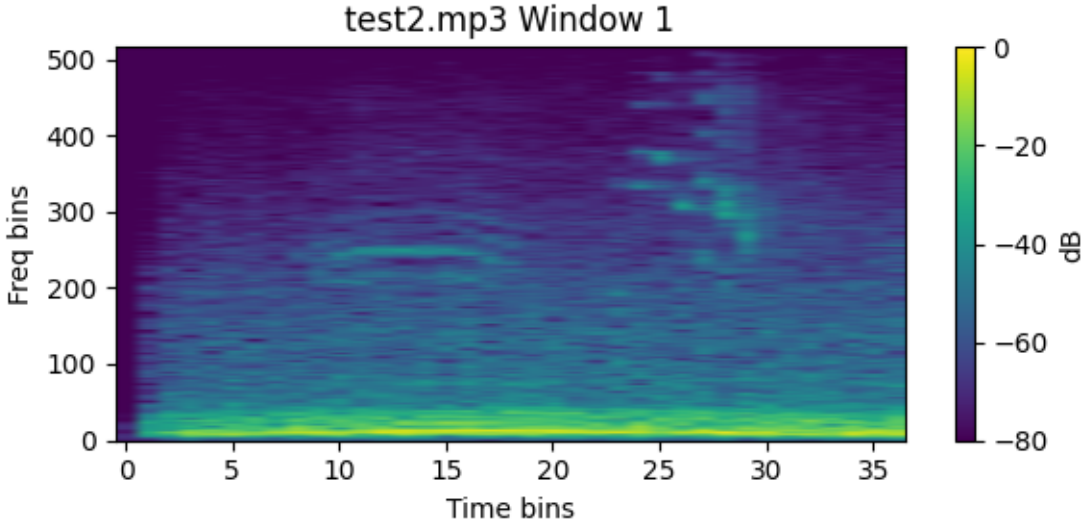


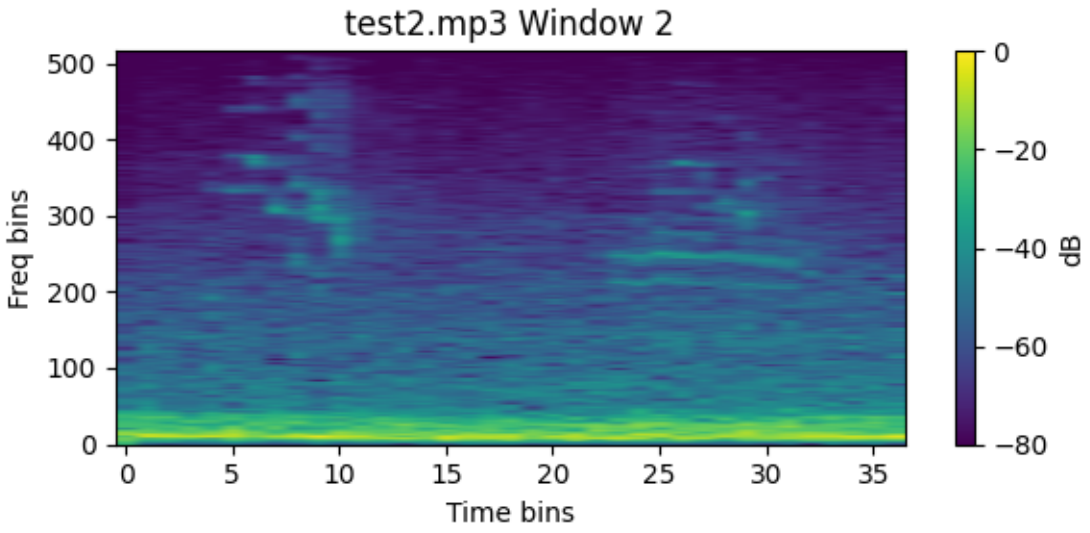
[27]:
```
# 3. Processing each clip
def normalize_spectrograms(data):
    return (data - np.mean(data)) / (np.std(data) + 1e-8)

for path in sorted(glob.glob('test_birds/*.mp3')):
    clip = path.split('/')[-1]
    y, _ = librosa.load(path, sr=sr, mono=True, duration=dur)
    y = np.pad(y, (0, max(0, int(sr*dur)-len(y))), 'constant')[:int(sr*dur)]

    for i, start in enumerate(range(0, len(y)-win_n+1, hop_n), 1):
        seg = y[start:start+win_n]
        spec = librosa.feature.melspectrogram(
            y=seg, sr=sr, n_mels=freq_bins, n_fft=2048,
            hop_length=int((win_n-2048)/(min_t-1))
        )
        spec_db = librosa.power_to_db(spec, ref=np.max)
        spec_db = spec_db[:,:min_t] if spec_db.shape[1]>=min_t else np.
 ↪pad(spec_db, ((0,0),(0,min_t-spec_db.shape[1])), constant_values=spec_db.
 ↪min())
        plt.figure(figsize=(6,3))
        plt.imshow(spec_db, origin='lower', aspect='auto')
        plt.title(f"{clip} Window {i}")
        plt.xlabel('Time bins'); plt.ylabel('Freq bins'); plt.
 ↪colorbar(label='dB'); plt.tight_layout(); plt.show()
        # Added same normalization as the multi class of train data

    # 4. Predicting probabilities per window
    probs = []
    for start in range(0, len(y)-win_n+1, hop_n):
        seg = y[start:start+win_n]
        spec = librosa.feature.melspectrogram(y=seg, sr=sr, n_mels=freq_bins,␣
 ↪n_fft=2048,
```

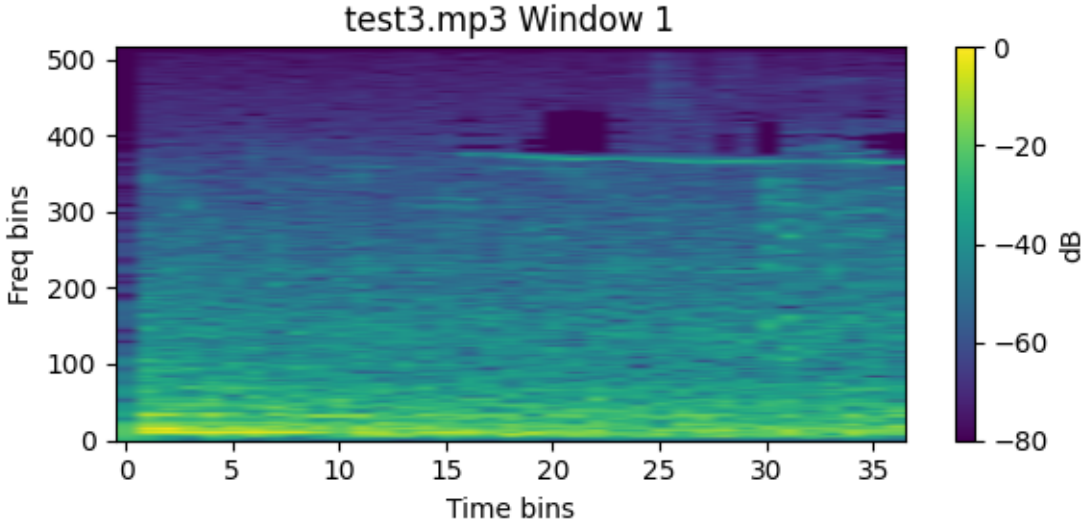
```
                                                hop_length=int((win_n-2048)/
↪(min_t-1)))
        spec_db = librosa.power_to_db(spec, ref=np.max)
        spec_db = spec_db[:,:min_t] if spec_db.shape[1]>=min_t else np.
↪pad(spec_db, ((0,0),(0,min_t-spec_db.shape[1])), constant_values=spec_db.
↪min())
        spec_db = normalize_spectrograms(spec_db)
        inp = spec_db[None,:,:,None]
        probs.append(m.predict(inp, verbose=0)[0])
    avg = np.mean(probs, axis=0)
    top3 = np.argsort(avg)[-3:][::-1]
    rows.append({
        'clip': clip,
        'top1_species': species_all[top3[0]], 'top1_prob': float(avg[top3[0]]),
        'top2_species': species_all[top3[1]], 'top2_prob': float(avg[top3[1]]),
        'top3_species': species_all[top3[2]], 'top3_prob': float(avg[top3[2]]),

    })
```

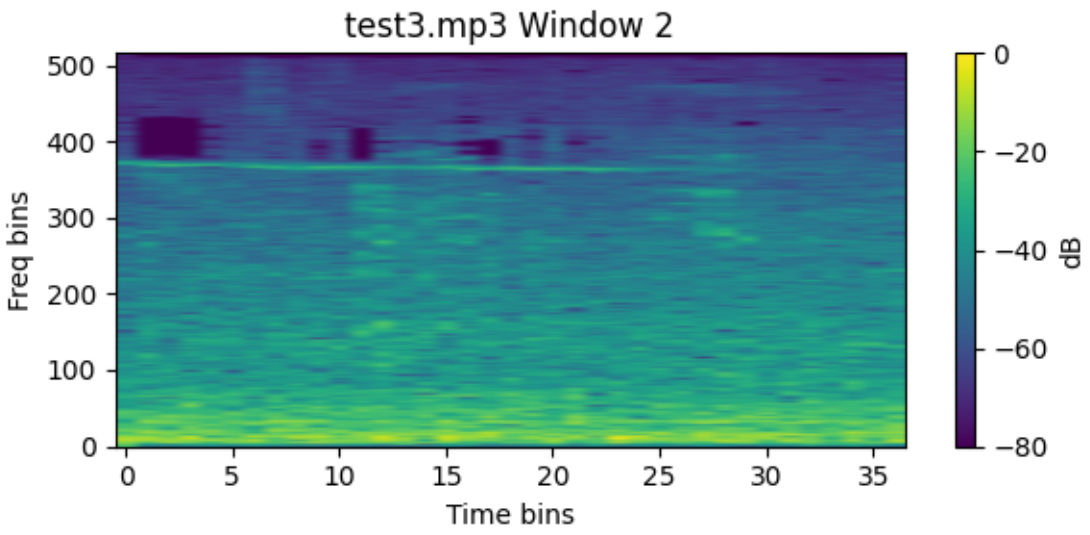



test1.mp3 Window 2
Freq bins
Time bins
dB
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80


test2.mp3 Window 1
Freq bins
Time bins
dB
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80


test2.mp3 Window 2
Freq bins
Time bins
dB
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80


test3.mp3 Window 1
Freq bins
Time bins
dB
0
100
200
300
400
500
0
5
10
15
20
25
30
35
0
−20
−40
−60
−80



[28]:
```
    # 5. Display top-3 table
df_rows = pd.DataFrame(rows)
print('=== External Test Data Top-3 Predictions:')
display(df_rows)
```

```
=== External Test Data Top-3 Predictions:
```

	clip	top1_species	top1_prob	top2_species	top2_prob	top3_species	top3_prob
0	test1.mp3	houspa	0.457843	bkcchi	0.316093	norfli	0.077289
1	test2.mp3	houspa	0.295434	bkcchi	0.247860	norfli	0.104093
2	test3.mp3	houspa	0.367825	bkcchi	0.282822	norfli	0.094744

[ ]: