DATA 5322 24SQ Statistical Machine Learning 2

Vatsal Dalal

5/11/2024

Bird Specie's Prediction Using Deep Learning

Abstract:

This assignment focuses on the neural network and its application to classify the bird species based on their sounds, focusing on species common in the Seattle area. The binary classification is done on the daejun and houfin. And doing multiclassification on 'amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3', 'norfli', 'rewbla', 'stejay', 'wesmea','whcspa' all the twelve bird species. For binary classification using CNN with dropout and basic neural network, however for multiclassification developing three neural network models. The first is CNN and CNN model with dropout and Basic Neural Network model. The dataset contains the spectrogram of 10 mp3 sound clips of various lengths for each of 12 bird species. Test data on 3 unlabeled mystery bird call clips along with test data featuring three unlabeled mystery bird call clips. Overall, the report underscores the significance of neural networks in advancing the field of bird species identification through sound analysis.

Introduction:

The study of bird behavior and population dynamics has lately gained popularity. Due to their quick response time to changing weather conditions, birds help us identify other species in the environment. Nevertheless, by utilizing the distinctive sounds of birds, current developments in machine learning have created opportunities for automating this procedure. This study uses audio samples of different bird species' calls to classify them using neural networks. The dataset comprises spectrograms derived from 10 MP3 sound clips for each of 12 bird species, along with three unlabeled mystery bird call clips for external testing. The main goal is to create robust neural network models that can identify different bird species from their distinctive sound patterns and make accurate predictions. To achieve this, the report discusses the design and implementation of binary and multi-class classification models, alongside a thorough exploration of various network structures and hyperparameters the possibilities and difficulties of utilizing neural networks to identify bird species are discussed in this work.

Theoretical Background:

Neural networks and deep learning are big topics in Computer Science and in the technology industry, they currently provide the best solutions to many problems in image recognition, speech recognition and natural language processing.

- **Basic Neural Network Architecture:**
  Basic Neural network consist of input layer, output layer hidden layers and activation function and weights.

Input layer: The input layer receives the data or input values. It does not perform any computations; it simply passes the input values to the next layer (hidden layer).

Hidden layer: Hidden layers are the core of the neural network, where the actual computations and transformations take place. Each node in a hidden layer receives weighted inputs from the nodes in the previous layer. These weighted inputs are summed, and an activation function is applied. Neural networks can have one or more hidden layers, depending on the complexity of the problem and the amount of data available.

Output Layer: The output layer is the final layer of the neural network, responsible for producing the desired output.

Activation Functions: These are the non-linear functions that are applied to the outputs of each hidden layer node, introducing non-linearity into the network. Common activation functions include ReLU, sigmoid, softmax and tanh.

- **Feedforward Neural Network:**

  Feedforward neural networks are the simple type of neural networks, where the data goes in one direction, from the input layer to the output layer. These networks are suitable for tasks such as image classification, regression, and pattern recognition.

- **Convolutional Neural Network (CNN):**

  Convolutional Neural Networks (CNNs) are images and recognition tasks. It uses the convolutional layers to extract features from the input data and pooling layers to reduce the dimensions of the feature maps.

Methodology:

- **Data Preprocessing:**

  The dataset is provided in h5 format and has already undergone preprocessing, so in order to prepare for binary classification, the labels and features of the two species, Daejun and Houfin, were chosen using a for loop. Features/np.max(features) was then used to normalize the features, converting them into a range between 0 and 1 that neural networks could understand. Additionally, features[..., np.newaxis]   By doing this, we are giving our features a new dimension. Additionally, data is prepared for model input using the train_test_split function.

  The same process is implemented for Multiclass Classification but instead of just two features we are selected all twelve features and data is prepared for input in model.

  But, unlabeled data for testing needs to be preprocessed, so we're developing the preprocess_clip function. It loads an audio file from the librosa library with a sr_rate of 22050. Next, librosa.feature.rms requires two parameters: n_ft = 2048 and hop length = 512. After computing the RMS energy, "Loud" segments are determined by applying a threshold equal to 50% of the maximum energy. It extracts the audio of the bird call for each "loud" segment, uses librosa.feature.melspectrogram to compute its mel-spectrogram, and resizes it to a fixed width of 343 frames. This is how the data is processed for unlabeled data: the spectrogram is then saved as a.npy file in the preprocessed_spectrograms files.

- **Binary Classification:**

  For binary classification in this work, I have used two custom neural network models – a simple neural network (NN) and a convolutional neural network (CNN) having an input shape of (256, 343, 1).

  The CNN model is designed with multiple ReLU activated convolutional layers that are followed by max-pooling layers to decrease the dimensionality. Dropout layers have also been added to mitigate over-fitting risks. The last layer has the activation function of sigmoid. To train the CNN model, Adam optimizer was employed with accuracy as well as binary cross-entropy loss being the evaluating metrics used in modeling.

  The NN consists of a flattened input, two dense layers activated through ReLU and a dropout layer for regularizing networks after each layer. The output layer uses sigmoid activation for binary classification just like in CNN. Moreover, Adam optimizer used as evaluation metrics like accuracy and binary cross-entropy loss.

- **Multiclass Classification:**
  Created three customized neural network models for multiclass classification: NN with dropout, CNN with dropout, and simple CNN. Additionally, the three models' input shapes are (256, 343, 1).

  The Basic CNN model has first flatten layer then multiple convolution layer with Relu as activation function followed by the maxpooling layer to reduce dimensionality and in ouput layer softmax is used as activation function. The model is trained with adam optimizer and sparse categorical crossentropy and accuracy for model evaluation.

  Another CNN model is same as the basic CNN but with adding multiple dropout layers in model. Dropout layers help prevent overfitting by randomly dropping a fraction of units during training.

  In the Neural network the first layer is flatten layer then three dense layer, and after all dense layer there is dropout layer is added to reduce the over fitting in model, and in the dense layer ReLu is used as activation function, and in output layer SoftMax is used for activation layer, the neural network is trained with adam optimizer and sparse categorical cross entropy and accuracy as a model evaluation metrics

- **Testing on Unlabeled Data:**
  To test on unlabeled audio clips, used the CNN model with dropout from multiclassification, first loaded the save model and compile the model, then used the preprocess spectrogram for prediction the bird species.

Computational Results:

- **Binary Classification:**

Models	Train Accuracy	Test Accuracy	Training Loss	Testing Loss
CNN with Dropout	1.0	1.0	0.0024	0.007
Simple NN	1.0	0.95	0.0011	0.077

Table 1: Binary Classification evaluation metrics

The binary classification evaluation metrics are displayed in the above table. Both models were trained using a batch size of 32 and 10 epochs. The table shows that the training accuracy of both models is 1.0, and the test accuracy is 1 and 0.95, respectively. While the training losses for Models 1 and 2 are 0.0011 and 0.0024, respectively, the testing losses for both models are similar 0.007.

➢ **Multiclass Classification:**

Models	Train Accuracy	Test Accuracy	Training Loss	Testing Loss
CNN Model	0.9962	0.6637	0.0170	2.94
CNN with Dropout	0.9722	0.7155	0.0806	2.10
Simple NN	0.6081	0.56	0.5957	1.44

Table 2: Evaluation metrics for multiclass classification

The multiclass classification evaluation metrics are shown in table 2. Basic CNN and simple NN models were trained using batch sizes of 16 and epochs of 10, while CNN with dropout model was trained on 20 epochs and their respective training accuracy, loss, and accuracy of training are 0.99, 0.97, and 0.6081 and losses are 0.010, 0.080, 0.5957 . The testing loss for each of the three models is 2.94, 2.10, and 1.20, and the testing accuracy is 0.66, 0.71, and 0.56.

➢ **Testing on Unlabeled Data:**

The most common predicted bird species for the tes1.mp3 files is the norfi, with probabilities of 0.9878; blujay is second, with probabilities of 0.9176; and stejay is third, with probabilities of 0.8045. For the test2.mp3 files, the most common birds are predicted to be norfi, with a probability of 0.9998, followed by blujay, with a value of 0.9899, and bikcchi, with a probability of 0.9990. According to the projected for the test3.mp3 file, the most common birds are blujay (probability of 0.8368), wesmea (0.9178), and norfi (0.8849).

Discussion:

➢ **Binary Classification:**

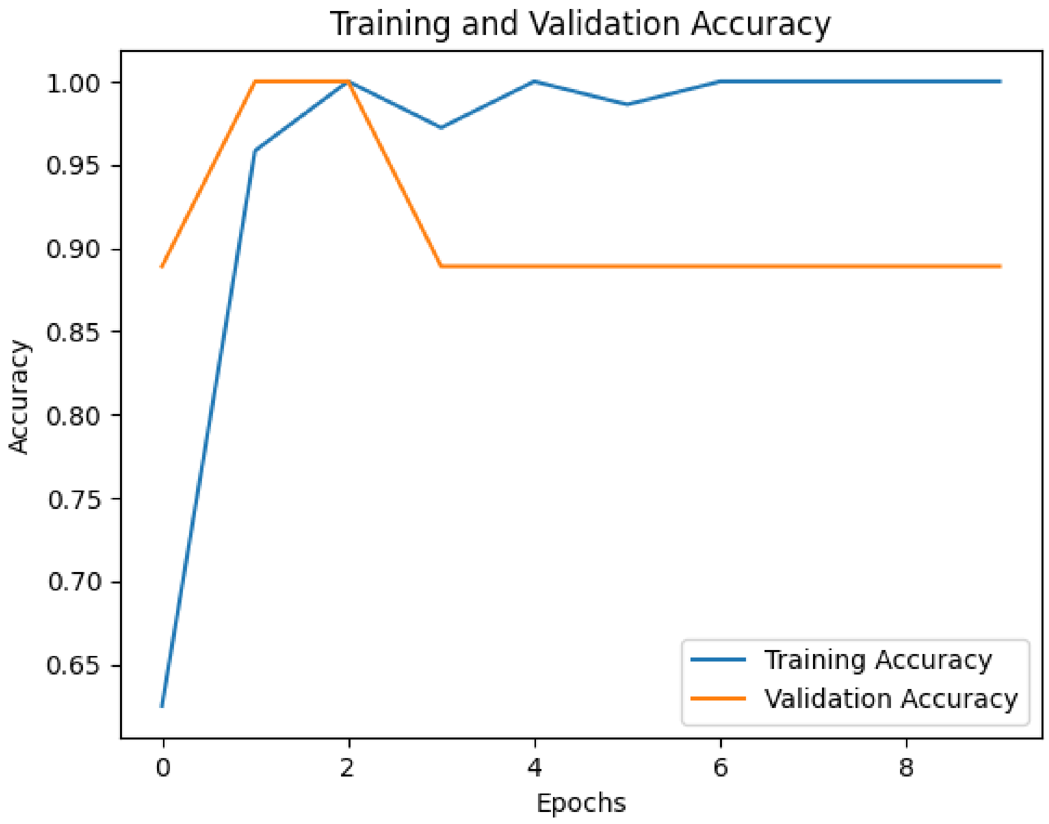

Fig 1: Training and validation accuracy for CNN Model

From fig 1: the training accuracy starts off relatively low and increases steadily over the course of training. This suggests that the model is learning from the training data. The validation accuracy also increases over time, but it does not increase as much as the training accuracy. This suggests that the model may be starting to overfit to the training data.

From the below fig 2: the training loss and validation loss graph we can see that both the training loss and the validation loss decrease over the course of training. This suggests that the model is learning from the training data and improving its performance. However, the training loss appears to be decreasing more quickly than the validation loss. This could be a sign that the model is overfitting to the training data.

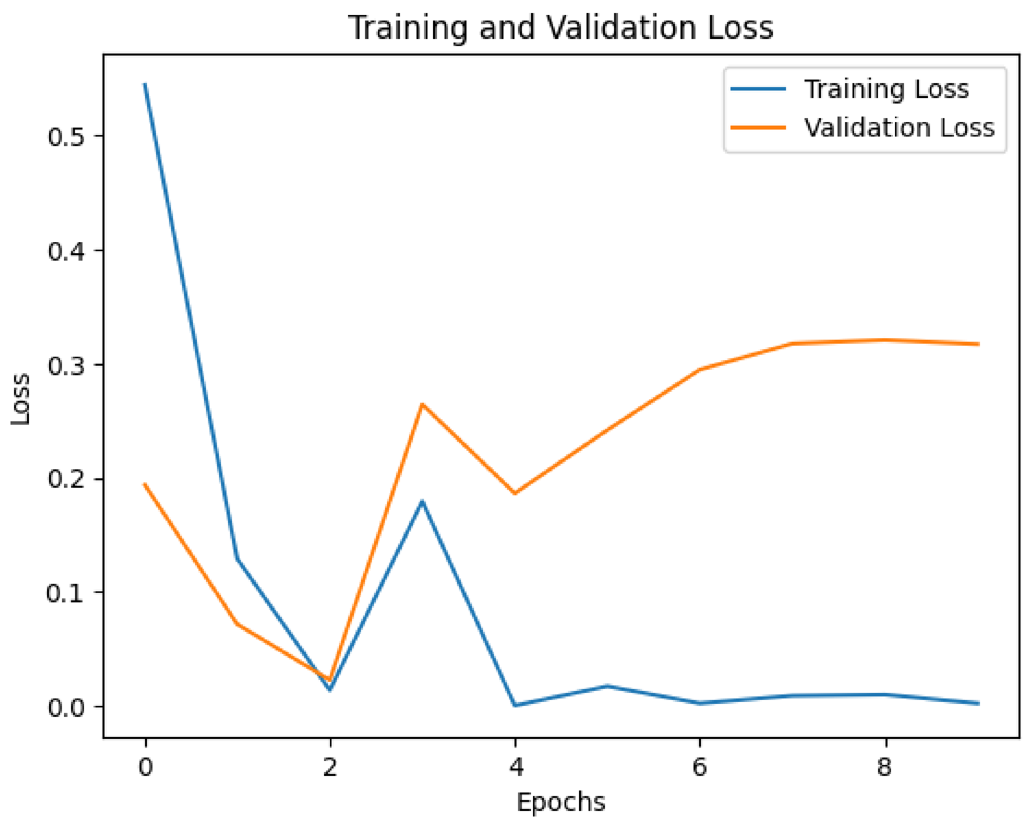

Fig 2: training and validation loss for CNN model

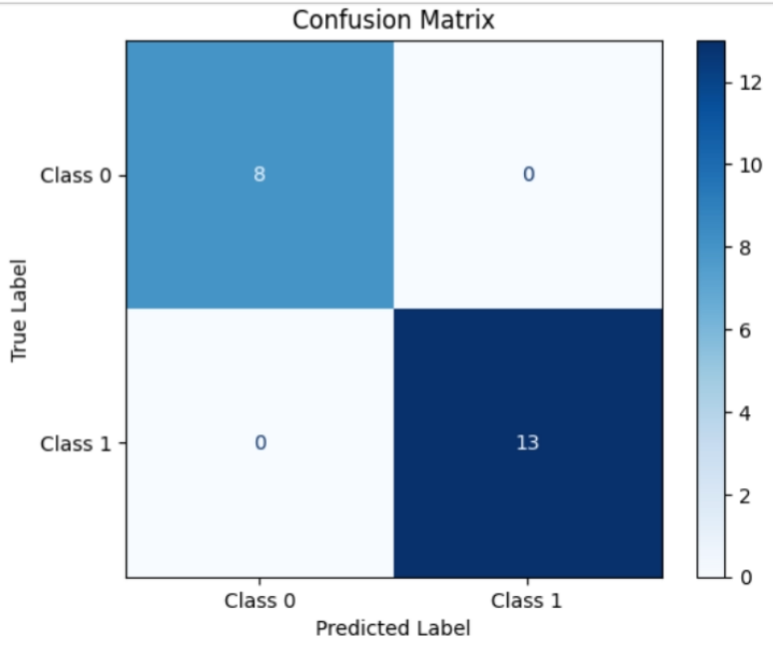

Fig 3: Confusion Matrix for CNN model

From the above fig 3: confusion matrix for CNN model shows that model has accurately predicted both birds species.

From the below fig 4: shows the training and accuracy of basic NN model, over the time validation accuracy remains constant and training accuracy increases and reached to 1.00.

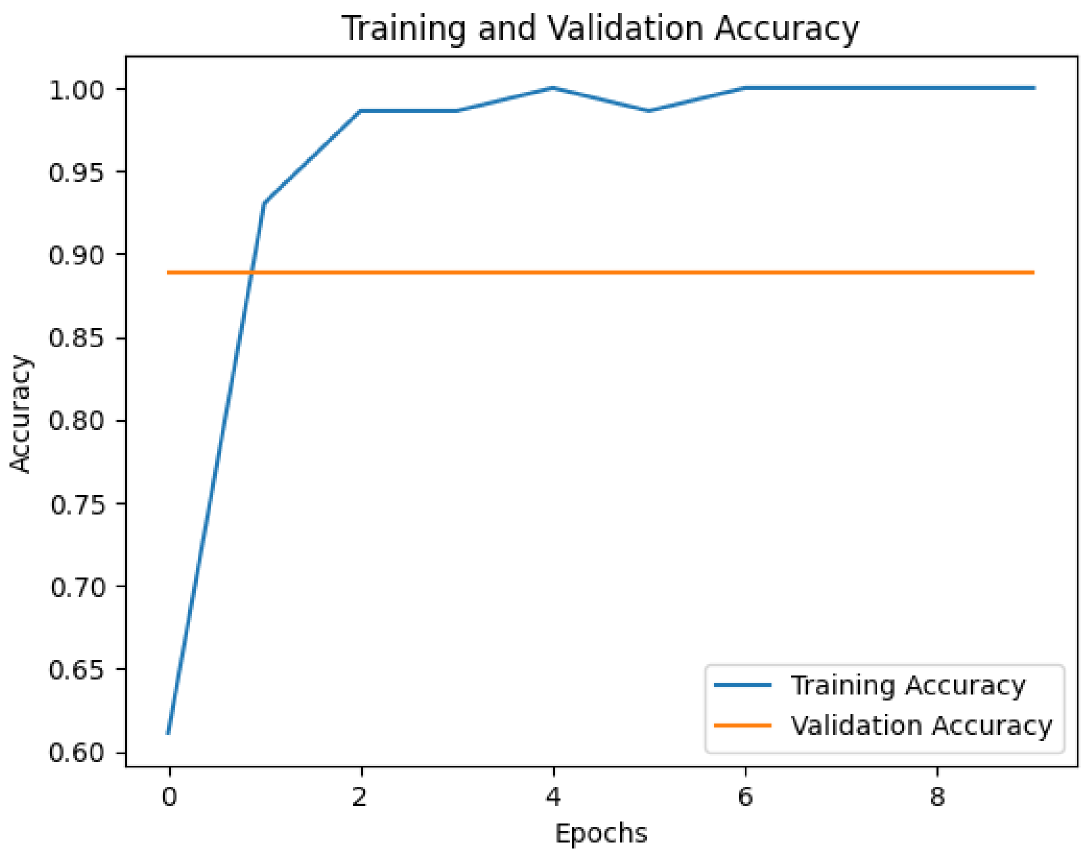

Fig 4: training and validation accuracy for basic NN model

From the below fig 5: show the training and validation loss for basic NN, training loss is decreasing while the validation loss start to decrease in beginning but in the end its again increasing.

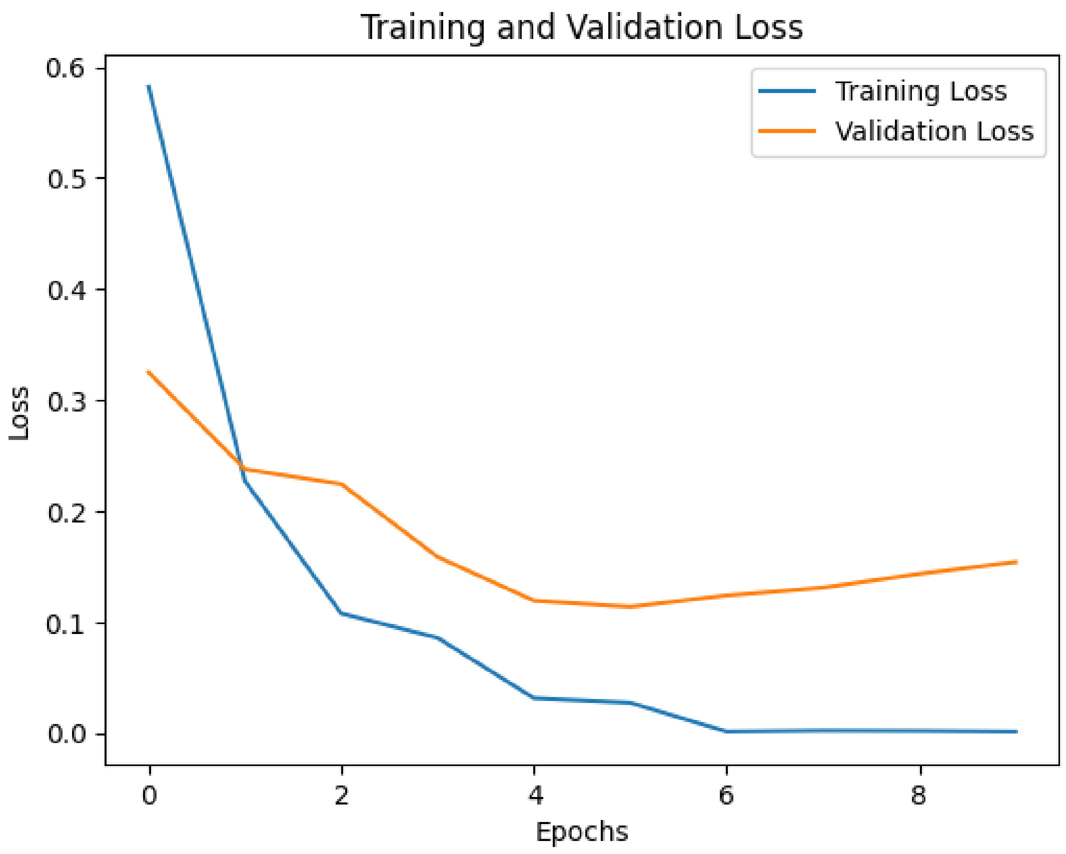

Fig 5: training and validation loss for basic NN model

➢ **Multiclass Classification:**

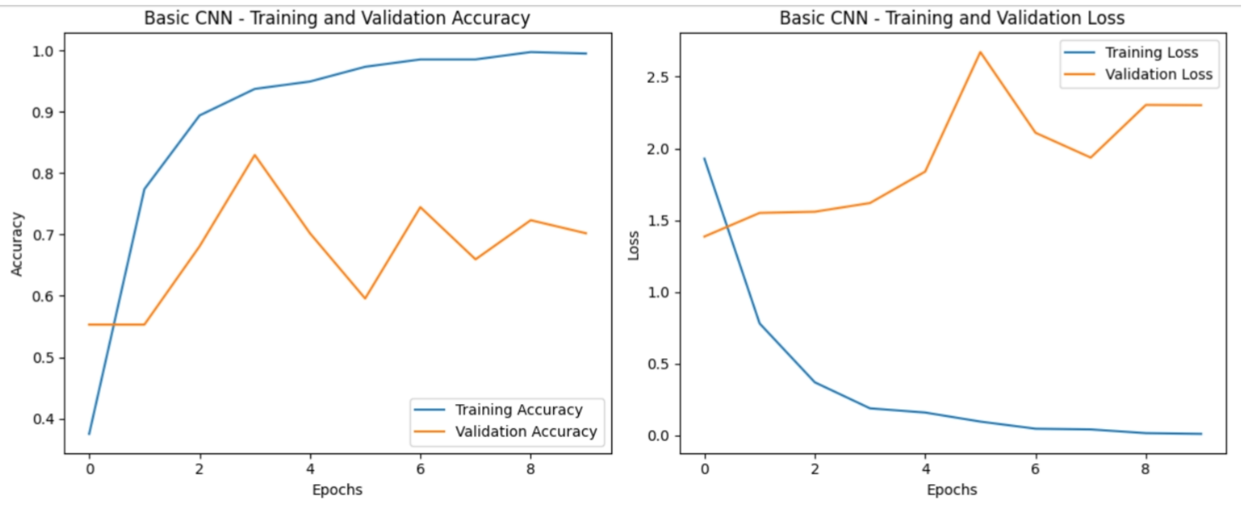

Fig 6: Training and validation graph for Basic CNN

From the fig 6 training accuracy increase and also the validation accuracy increases but fluctuates more than the training accuracy. This could be a sign of overfitting, however we can clearly see in loss graph model is overfitting.

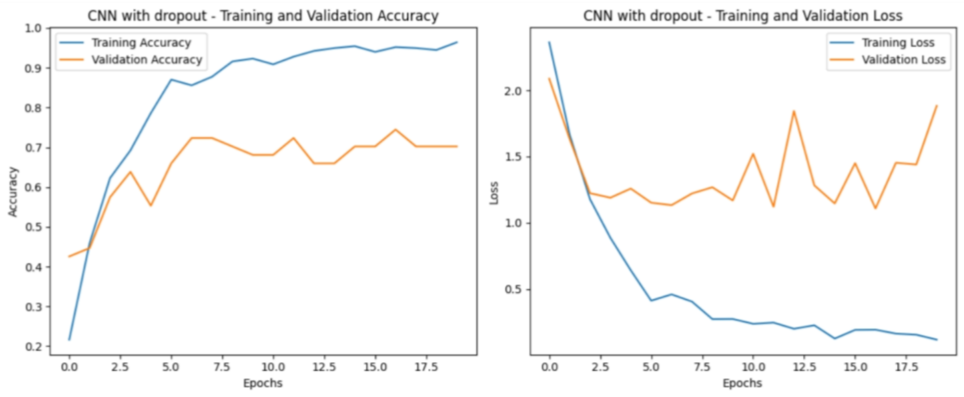

Fig 7: training and validation plot for CNN with dropout model

From the fig 7: the training and validation accuracy is increasing model is performing well on validation set and but after some time validation accuracy remains constant, furthermore training loss is decreasing while validation loss is also increased in starting but after 10 epochs it unstable it's in increasing and decreasing manner.

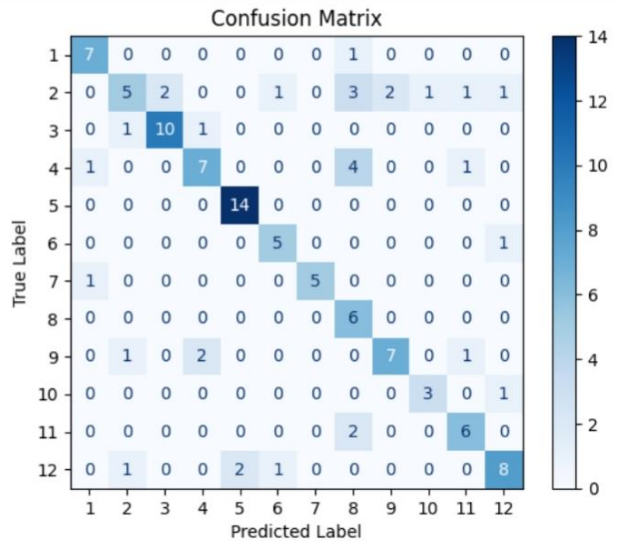

Fig 8: confusion matrix for CNN model with dropout for multiclass classification

Figure 8 presents confusion matrix which represents the model's prediction accuracy among different labels. Model accurately predicts label 5 (daejun) and label 3 (bkcchi) most of the time. Meanwhile, there are cases where the model predicts other labels wrongly. Particularly, misclassification often happens with respect to label 4 (blujay) as well as label 2 (barswa). Moreover, similar error is made in trying to distinguish it from other misclassified rows like row for label 8 (norfli).

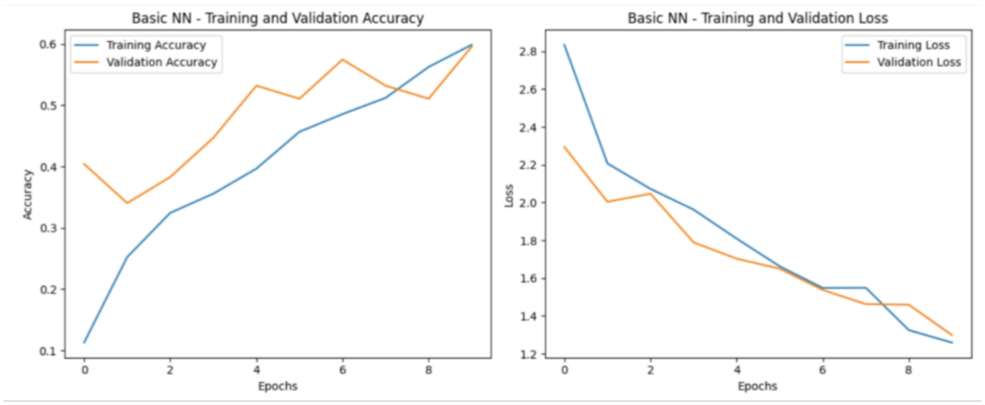

Fig 9: Basic NN validation and training plots

From the fig 9 we can say that model is performing well on the data there might be chance of overfitting as well and for loss model is performing well adapting the unseen data on validation set but accuracy for test data is 0.58.

What limitations did you run into in this homework? How long did it take to train the models?

The neural network model takes a lot of time to consuming for binary classification it takes around 15 minutes to run the model for 10 epochs, while in multiclass classification the model takes about 25 minutes to run.

Which species proved the most challenging to predict, or did any get confused for one another frequently? Listen to the bird calls and look at the spectrograms. Is there any characteristic of the bird call that makes this so?

Based on the confusion matrix in the multiclass CNN with dropout model (fig. 8), the model predicts the correct bird most of the time. However, for the birds 'norfli','rewbla','stejay', 'wesmea', and 'whcspa', the model makes mistakes and predicts the wrong bird, barswa. I believe this is because the sounds of these birds are similar, so there may be some overlap in the sounds, which is the reason the model is inaccurate.

What other models could we have used to perform this task? Why would a neural network make sense for this application?

There are some pretrained model like Resnet and VGG16, RESNET and Mobile Net we can convert the audio files into spectrograms and use the pre trained model and also there is AudioSet dataset is provided by google we can also use this. And neural networks speacially CNN is good for pattern detection, adaptability, Temporal Hierarchies and CNNs can effectively operate on spectrogram-like data, where convolutional filters can capture frequency patterns across time.

Conclusion:

In summary, this research looks at the use of artificial neural networks for bird species recognition using their unique sound patterns. We explore several neural network architectures and designs like simple neural networks, convolutional neural networks (CNNs), and neural networks with dropout layers in this report by conducting binary classification between deajun and houfin species; as well as 12-way multiclassification among different bird species common in Seattle.

The computational results indicate that these models are effective, with both binary and multiclass classification models achieving high training and testing accuracies, particularly in the binary classification task. The findings show that despite encountering such challenges as overfitting or different levels of accuracy across various bird species, the developed neural network models were able to identify bird species based on spectrogram data quite accurately. Moreover, the efficacy of those models was also confirmed by testing on unlabeled data where most of the mystery bird call clips were predicted correctly.

REFERENCES:

1. P. Anusha and K. ManiSai, "Bird Species Classification Using Deep Learning," 2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP), Hyderabad, India, 2022, pp. 1-5, doi: 10.1109/ICICCSP53532.2022.9862344. keywords: {Training;Deep learning;Tensors;Shape;Image color analysis;Convolution;Linux;Autograph;Caltech-UCSD;grey scale pixels;Tensorflow}, https://ieeexplore.ieee.org/document/9862344
2. https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b877
3. https://www.quora.com/Why-does-CNN-work-also-well-for-audio-data-besides-image-data-What-kind-of-features-do-they-try-to-extract-from-audio-data-that-are-useful-for-solving-problems-like-audio-classification
4. https://librosa.org/doc/latest/tutorial.html
5. https://www.tensorflow.org/guide/keras

Appendix:

```
import h5py

import numpy as np

import numpy as np

from sklearn.model_selection import train_test_split

import tensorflow as tf
```

```
from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

f = h5py.File('./spectrograms.h5', 'r')

species_keys = list(f.keys())

species_keys
list(f.items())
```

Binary Classification:

```
# Get data for "daejun" and "houfin"

data_species_1 = f["daejun"][:]

data_species_2 = f["houfin"][:]


features = []

labels = []


# Add data for "daejun" with label 1

for i in range(data_species_1.shape[2]):

    features.append(data_species_1[:, :, i])  # Add spectrograms

    labels.append(1)  # Label for "daejun"


# Add data for "houfin" with label 0

for i in range(data_species_2.shape[2]):

    features.append(data_species_2[:, :, i])  # Add spectrograms

    labels.append(0)  # Label for "houfin"


# Convert features and labels to numpy arrays
```

```
features = np.array(features)

labels = np.array(labels)

features


features = features / np.max(features)  # Normalize to [0, 1]

features

features = features[..., np.newaxis]  # Adding the channel dimension

features

X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2, random_state=42)


# Display the shapes to ensure the split worked correctly

print("X_train shape:", X_train.shape)

print("y_train shape:", y_train.shape)

print("X_test shape:", X_test.shape)

print("y_test shape:", y_test.shape)

cnn_model = keras.Sequential([

    # First convolutional layer

    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256,
343, 1)),  # Input shape is (256, 343, 1)

    layers.MaxPooling2D((2, 2)),  # First pooling layer

    layers.Dropout(0.4),

    # Second convolutional layer

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),  # Second pooling layer


    # Third convolutional layer

    layers.Conv2D(128, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),  # Third pooling layer
    layers.Dropout(0.4),
    # Flatten the output for dense layers
    layers.Flatten(),

    # Fully connected dense layer
    layers.Dense(128, activation='relu'),  # Relu activation
    layers.Dropout(0.5),  # Dropout with a rate of 50%

    # Output layer with sigmoid activation for binary classification
    layers.Dense(1, activation='sigmoid')
])
cnn_model.compile(
    optimizer='adam',  # Adam optimizer
    loss='binary_crossentropy',  # Binary cross-entropy loss for
binary classification
    metrics=['accuracy']  # Using accuracy as a metric
)
history = cnn_model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

test_loss, test_accuracy = cnn_model.evaluate(X_test, y_test)


print("Test Loss:", test_loss)

print("Test Accuracy:", test_accuracy)

y_pred = (cnn_model.predict(X_test) > 0.3).astype("int32")  # Assuming
binary classification


# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


# Plot confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=['Class 0', 'Class 1'])

disp.plot(cmap=plt.cm.Blues, values_format='d')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

simple_nn = keras.Sequential([

    # Input layer (Flatten to ensure compatibility with dense layers)

    layers.Flatten(input_shape=(256, 343, 1)),  # Flatten from (256,
343, 1) to (256*343)
```

```
    # First dense layer with ReLU activation
    layers.Dense(128, activation='relu'),  # Fully connected layer

    # Dropout to reduce overfitting
    layers.Dropout(0.3),  # Dropout with a rate of 30%

    # Second dense layer with ReLU activation
    layers.Dense(64, activation='relu'),  # Second dense layer

    # Dropout after dense layer
    layers.Dropout(0.3),  # Dropout with a rate of 30%

    # Output layer with sigmoid activation for binary classification
    layers.Dense(1, activation='sigmoid')  # Sigmoid output for binary
classification
])
simple_nn.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
history = simple_nn.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
test_loss, test_accuracy = simple_nn.evaluate(X_test, y_test)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

Multiclassification:

```
features = []
labels = []

# Assign each species a unique label
species_keys = list(f.keys())
species_labels = {species: idx for idx, species in enumerate(species_keys)}

# Populate the features and labels with data from all species
for species, label in species_labels.items():
    data_species = f[species][:]
    for i in range(data_species.shape[2]):
        features.append(data_species[:, :, i])
        labels.append(label)
```

```
# Convert features and labels to numpy arrays

features = np.array(features)

labels = np.array(labels)


# Normalize the features

features = features / np.max(features)


# Reshape features to add the channel dimension (e.g., grayscale images have
1 channel)

features = features[..., np.newaxis]


# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2, random_state=42)


print("X_train shape:", X_train.shape)

print("y_train shape:", y_train.shape)

print("X_test shape:", X_test.shape)

print("y_test shape:", y_test.shape)

num_classes = len(species_labels)


# First Model: Basic CNN (without Dropout layers)

cnn_model_basic = keras.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 343, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),

    layers.Dense(128, activation='relu'),

    layers.Dense(num_classes, activation='softmax')

])


cnn_model_basic.compile(optimizer='adam',

                        loss='sparse_categorical_crossentropy',

                        metrics=['accuracy'])


# Fit the Basic CNN

history_basic = cnn_model_basic.fit(X_train, y_train, epochs=10,
batch_size=16, validation_split=0.1)


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


# Plot training and validation accuracy

plt.subplot(1, 2, 1)

plt.plot(history_basic.history['accuracy'], label='Training Accuracy')

plt.plot(history_basic.history['val_accuracy'], label='Validation Accuracy')

plt.title('Basic CNN - Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()


# Plot training and validation loss

plt.subplot(1, 2, 2)

plt.plot(history_basic.history['loss'], label='Training Loss')

plt.plot(history_basic.history['val_loss'], label='Validation Loss')

plt.title('Basic CNN - Training and Validation Loss')

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
test_loss, test_accuracy = cnn_model_basic.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
cnn_model_dropout = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 343, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

cnn_model_dropout.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])

# Fit the CNN with Dropout
```

```
history_dropout = cnn_model_dropout.fit(X_train, y_train, epochs=20,
batch_size=16, validation_split=0.1)

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


# Plot training and validation accuracy

plt.subplot(1, 2, 1)

plt.plot(history_dropout.history['accuracy'], label='Training Accuracy')

plt.plot(history_dropout.history['val_accuracy'], label='Validation
Accuracy')

plt.title('CNN with dropout - Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()


# Plot training and validation loss

plt.subplot(1, 2, 2)

plt.plot(history_dropout.history['loss'], label='Training Loss')

plt.plot(history_dropout.history['val_loss'], label='Validation Loss')

plt.title('CNN with dropout - Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()


plt.tight_layout()

plt.show()

test_loss, test_accuracy = cnn_model_dropout.evaluate(X_test, y_test)

print("Test Loss:", test_loss)

print("Test Accuracy:", test_accuracy)

# Make predictions on the test data

y_pred = np.argmax(cnn_model_dropout.predict(X_test), axis=1)  # Get the
index of the highest probability class
```

```
# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


# Plot confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=[1,2,3,4,5,6,7,8,9,10,11,12])

disp.plot(cmap=plt.cm.Blues, values_format='d')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

dense_model_dropout = keras.Sequential([

    layers.Flatten(input_shape=(256, 343, 1)),  # Flattening the input image

    layers.Dense(128, activation='relu'),  # First dense layer

    layers.Dropout(0.5),  # Dropout layer

    layers.Dense(128, activation='relu'),  # Second dense layer

    layers.Dropout(0.5),  # Another Dropout

    layers.Dense(num_classes, activation='softmax')  # Output layer

])


dense_model_dropout.compile(optimizer='adam',

                            loss='sparse_categorical_crossentropy',

                            metrics=['accuracy'])


# Fit the Dense Network with Dropout

history_dense = dense_model_dropout.fit(X_train, y_train, epochs=10,
batch_size=16, validation_split=0.1)

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

```
# Plot training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history_dense.history['accuracy'], label='Training Accuracy')
plt.plot(history_dense.history['val_accuracy'], label='Validation Accuracy')
plt.title('Basic NN - Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history_dense.history['loss'], label='Training Loss')
plt.plot(history_dense.history['val_loss'], label='Validation Loss')
plt.title('Basic NN - Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
test_loss, test_accuracy = dense_model_dropout.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
cnn_model_dropout.save("cnn_model_dropout.h5")
```

Testing on Unlabeled data:

```
import librosa
import os

from tensorflow.keras.models import load_model

def preprocess_clip(file_path, output_dir, sample_rate=22050,
bird_call_window_size=2, n_fft=2048, hop_length=512):


    y, sr = librosa.load(file_path, sr=sample_rate)


    energy = librosa.feature.rms(y=y, frame_length=n_fft,
hop_length=hop_length)




    loud_indexes = np.where(energy > np.max(energy) * 0.5)[1]


    for loud_idx in loud_indexes:

        # Compute the start and end times of the bird call window

        start_time = loud_idx * hop_length / sr

        end_time = start_time + bird_call_window_size


        # Extract the bird call audio

        bird_call_audio = y[int(start_time * sr):int(end_time * sr)]


        # Calculate the mel-spectrogram

        spectrogram = librosa.feature.melspectrogram(y=bird_call_audio,
sr=sr, n_fft=n_fft, hop_length=hop_length,
                                                                     n_mels=36, fmax=8000)


        # Resize the spectrogram to have a fixed width of 343 frames

        if spectrogram.shape[1] < 343:

            padding = np.zeros((36, 343 - spectrogram.shape[1]))

            spectrogram = np.hstack((spectrogram, padding))

        elif spectrogram.shape[1] > 343:
            spectrogram = spectrogram[:, :343]

        # Save the spectrogram
        filename = os.path.splitext(os.path.basename(file_path))[0] +
f'_window_{start_time:.2f}.npy'
        np.save(os.path.join(output_dir, filename), spectrogram)

# Directory containing the sound clips
sound_clips_dir = './test_birds/'

# Directory to save preprocessed spectrograms
output_dir = './preprocessed_spectrograms/'

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Iterate over sound clips in the directory
for filename in os.listdir(sound_clips_dir):
    if filename.endswith('.mp3'):
        file_path = os.path.join(sound_clips_dir, filename)
        preprocess_clip(file_path, output_dir)

print("Preprocessing completed.")
model = load_model('./cnn_model_dropout.h5')
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
print(model.summary())
# Define the mapping of class indices to bird species names
class_to_species = {
    1: 'amecro', 2: 'barswa', 3: 'bkcchi', 4: 'blujay', 5: 'daejun', 6:
'houfin',
    7: 'mallar3', 8: 'norfli', 9: 'rewbla', 10: 'stejay', 11: 'wesmea', 12:
'whcspa'

}


preprocessed_spectrograms_dir = './preprocessed_spectrograms/'


# List to store predictions for each spectrogram

predictions = []

file_names = []


# Iterate over preprocessed spectrograms in the directory

for filename in os.listdir(preprocessed_spectrograms_dir):

    if filename.endswith('.npy'):

        file_path = os.path.join(preprocessed_spectrograms_dir, filename)

        spectrogram = np.load(file_path)

        spectrogram = spectrogram.flatten()

        spectrogram = np.pad(spectrogram, (0, 256*343 - spectrogram.size),
mode='constant')

        spectrogram = np.reshape(spectrogram, (1, 256, 343, 1))  # Reshape to
match model's input shape

        prediction = model.predict(spectrogram)

        predictions.append(prediction)

        file_names.append(filename.split('.')[0])  # Get the file name
without the extension


# Convert predictions to a numpy array

predictions = np.array(predictions)


# Iterate over predictions and file names

for i, (prediction, file_name) in enumerate(zip(predictions, file_names)):

    # Get the original label from the file name
    original_label = file_name.split('_window_')[0]
    print(f"Original Label: {original_label}")

    # Get the top 5 predicted class indices and probabilities
    top_indices = prediction[0].argsort()[-5:][::-1]
    top_probs = prediction[0][top_indices]

    # Print the top 5 predictions
    for j, (index, prob) in enumerate(zip(top_indices, top_probs)):
        species = class_to_species[index]
        print(f"Top {j+1} Predicted Bird Species: {species} - Class: {index}
- Probability: {prob:.4f}")

    print()
```