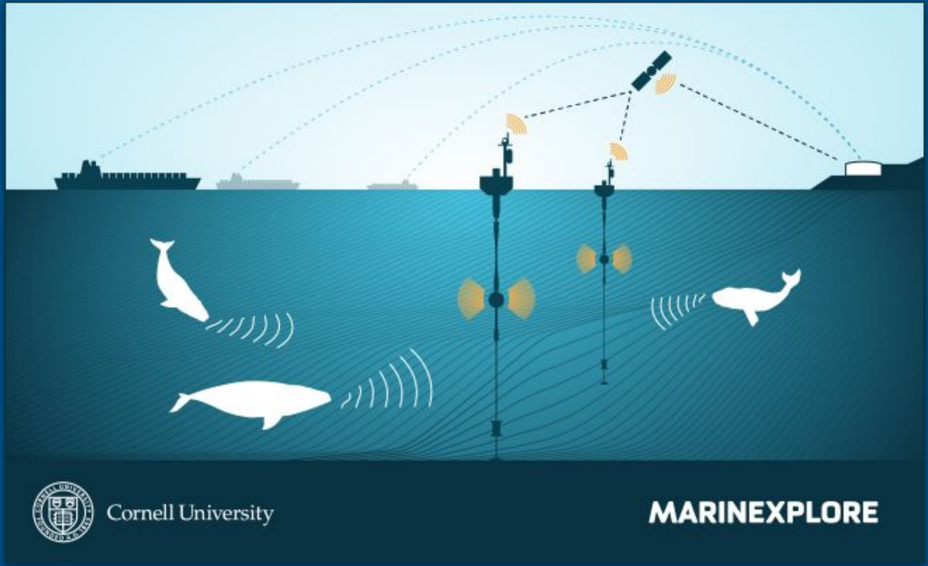
Whale Vocalization Detection using Deep Learning

Christopher Strater

Daniel Cwynar

Sutikshan Bandharu

Github Repository:
https://github.com/javadahut/Final-Github-Repository



1. Motivation & Introduction
2. Dataset Overview & Spectrograms
3. Model Architectures
4. Results
5. Applications & Future Work
6. Conclusion

Motivation & Introduction

- Whales are acoustic dependent animals and heavily rely on sound for communication, navigation, and survival
- Expanding shipping around the world increases the risk of ship-whale collisions
- Passive Acoustic Monitoring (PAM) has emerged for detecting whale vocalizations using hydrophones
- Our goal is to reduce the likelihood of fatal encounters for whales and other marine life

- Our project builds off of the Kaggle Whale Detection Challenge from Cornell University and Marinexplore: https://www.kaggle.com/competitions/whale-detection-challenge
- Benchmark 5 total models using AUROC, AUPRC, and classification accuracy, targeting a minimum AUROC of 0.98

Methodology for This Project

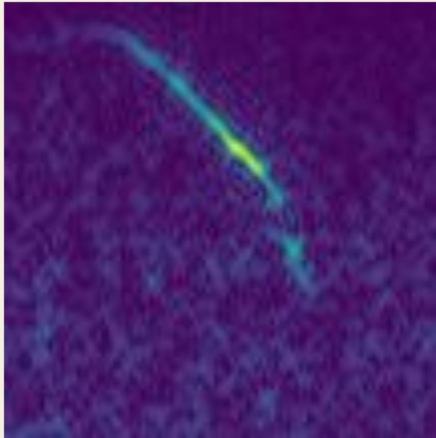
1. Converting raw audio waveforms into spectrogram representations suitable for input into our models
2. Defining five model architectures (CNN, Inception-based CNN, CenterLossSimple, EfficientNet, and AST transformer)
3. Training and evaluating these algorithms on our whale call dataset.

Dataset Overview & Spectrograms

- **30,000 training**, 54,503 testing
- Highly imbalanced dataset 10-15% True labels
- 80% train, 10% validation, 10% test
- **Short-Time Fourier Transform (STFT)**: 2D image representing how acoustic energy is distributed over time and frequency
- **Mel Spectrogram**: better represent how frequency content is perceived by humans and marine species
- Process: .aiff files read and normalized -> STFT & Mel spectrograms generated -> resized and stored as RGB tensors -> saved as NumPy format or Torch Tensors

$$X[k,m] = \left| \sum_{n=mK_F}^{K_F(m+1)-1} x[n]\, e^{-j\frac{2\pi nk}{K_F}} \right|$$

clip_name	label
train1.aiff	0
train2.aiff	0
train3.aiff	0
train4.aiff	0
train5.aiff	0
train6.aiff	1
train7.aiff	1
train8.aiff	0
train9.aiff	1
train10.aiff	0



Model Architectures

Custom CNN (108×108)

- Used as the baseline CNN model
- Input: 108×108 Mel-spectrograms
- 5 total Convolutional layers 3x3 kernels and ReLU
- Incorporates Max Pooling and Batch Normalization
- Filter depth progression: Outputs: 8 -> 16 -> 32 -> 64 -> 64 channels
- FC: Linear(256 neurons) -> dropout -> Linear(32) Binary Output
- Fast to train and has a good receptive field

```
def cnn_108x108(self):
    self.cnn = nn.Sequential(
        nn.Conv2d(1, 8, 3, stride=(1,1), padding=(1,1)),
        nn.BatchNorm2d(8),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(8, 16, 3, stride=(1,1), padding=(1,1)),
        nn.BatchNorm2d(16),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(16, 32, 3, stride=(1,1), padding=(0,0)),
        nn.Dropout2d(),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(32, 64, 3, stride=(1,1), padding=(1,1)),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(64, 64, 3, stride=(3,3), padding=(0,0)),
        nn.Dropout2d(),
        nn.ReLU(inplace=True)
    )
    self.classifier = nn.Sequential(
        nn.Linear(256, 32),
        nn.Dropout(),
        nn.Linear(32, 16),
        nn.Linear(16, 2)
    )
```

InceptionV1 CNN and Variants

- Based on GoogLeNet InceptionV1
- Learns multi-scale features with branches of different filter sizes (1×1, 3×3, 5×5)
- Balanced trade off between speed, and receptive field
  - **inceptionModuleV1_108x108** – baseline for full-resolution inputs
  - **inceptionModuleV1_75x45** – optimized for lower-resolution spectrograms
  - **inceptionTwoModulesV1_75x45** – stacks two inception blocks sequentially
  - **inceptionV1_modularized** – parameterized depth via the -nils argument
  - **inceptionV1_modularized_mnist** – minimal version for small-scale prototypes

```python
def inceptionModuleV1_108x108(self):
    self.root = nn.Sequential(
        nn.Conv2d(1, 64, 3, stride=(1,1), padding=(1,1)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(64, 192, 3, stride=(1,1), padding=(1,1)),
        nn.BatchNorm2d(192),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2))
    )
    self.b_1x1 = nn.Sequential(
        nn.Conv2d(192, 64, 1, stride=(1,1), padding=(0,0)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True)
    )
    self.b_3x3 = nn.Sequential(
        nn.Conv2d(192, 96, 1, stride=(1,1), padding=(0,0)),
        nn.BatchNorm2d(96),
        nn.ReLU(inplace=True),
        nn.Conv2d(96, 128, 3, stride=(1,1), padding=(1,1)),
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True)
    )
    self.b_5x5 = nn.Sequential(
        nn.Conv2d(192, 16, 1, stride=(1,1), padding=(0,0)),
        nn.BatchNorm2d(16),
        nn.ReLU(inplace=True),
        nn.Conv2d(16, 32, 5, stride=(1,1), padding=(2,2)),
        nn.BatchNorm2d(32),
        nn.ReLU(inplace=True)
    )
    self.b_pool = nn.Sequential(
        nn.MaxPool2d((3,3), stride=(1,1), padding=(1,1)),
        nn.Conv2d(192, 32, 1, stride=(1,1), padding=(0,0)),
        nn.BatchNorm2d(32),
        nn.ReLU(inplace=True)
    )
    self.redux = nn.Sequential(
        nn.Conv2d(256, 64, 2, stride=(1,1), padding=(1,1)),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(64, 32, 1, stride=(1,1), padding=(0,0)),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(32, 16, 1, stride=(1,1), padding=(0,0)),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((3,3), stride=(2,2), padding=(0,0)),
        nn.Conv2d(16, 4, 1, stride=(1,1), padding=(0,0)),
        nn.ReLU(inplace=True)
    )
    self.classifier = nn.Sequential(
        nn.Linear(36, 2)
    )
```

CenterLossSimple

- Incorporates Center Loss (Pulls features of the same class toward a learned centroid)
- Adds two linear layers
  - One for computing distances to class centers
  - One for final classification
- Improves intra-class compactness and reduces confusion in noisy data

```
def centerlossSimple(self, nEmbed, nClasses):
    self.centroids = torch.from_numpy(0.01*np.random.randn(nEmbed, nClasses)).cuda(0)
    self.root = nn.Sequential(
        nn.Conv2d(1, 32, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.Conv2d(32, 32, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(32, 64, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.Conv2d(64, 64, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2)),
        nn.Conv2d(64, 128, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.Conv2d(128, 128, 5, stride=(1,1), padding=(2,2), bias=True),
        nn.ReLU(inplace=True),
        nn.MaxPool2d((2,2), stride=(2,2))
    )
    self.latent = nn.Linear(1152, nEmbed, bias=True)
    self.logits = nn.Linear(nEmbed, nClasses, bias=False)
```

EfficientNetV2-S (224x224)

- Pretrained CNN and fine tuned
- Inputs 224x224 RGB Mel-Spectrograms
- Utilizes conventional MBConv block (expand first then shrink)
- Perfect for real-time deployment

```python
def efficientnetV2_S(self):
    import timm
    model = timm.create_model('tf_efficientnetv2_s', pretrained=True)
    # Adapt the first convolution to accept one-channel input if needed.
    if model.conv_stem.in_channels != 1:
        new_conv = nn.Conv2d(
            1,
            model.conv_stem.out_channels,
            kernel_size=model.conv_stem.kernel_size,
            stride=model.conv_stem.stride,
            padding=model.conv_stem.padding,
            bias=False
        )
        new_conv.weight.data = model.conv_stem.weight.data.mean(dim=1, keepdim=True)
        model.conv_stem = new_conv
    in_features = model.classifier.in_features
    model.classifier = nn.Linear(in_features, 2)
    self.model = model
```

Audio Spectrogram Transformer (AST)

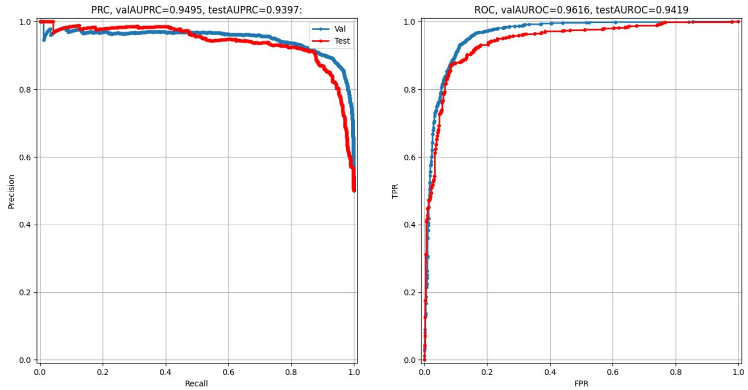
- Pretrained Transformer on AudioSet and fine tuned
- TorchAudio to load audio as waveform
- Data passed through ASTFeatureExtractor to get spectrograms
- Model splits spectrograms into sequences of patches
- Self attention great for long range dependencies

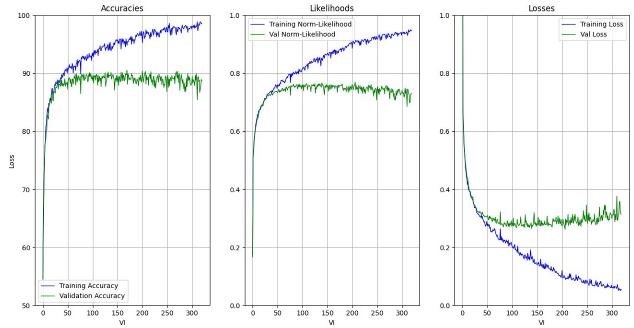
```
class SpectrogramDataset(Dataset):
    def __init__(self, data_path, labels_path):

        self.df = pd.read_csv(labels_path)

        # Create data and labels
        self.data = [os.path.join(data_path, fname) for fname in self.df["clip_name"]]
        self.labels = torch.tensor(self.df["label"].values).long()

        # AST features
        self.feature_extractor = ASTFeatureExtractor()


    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):

        file_path = self.data[idx]
        waveform, sample_rate = torchaudio.load(file_path)

        # Resample to 16k
        waveform = torchaudio.functional.resample(waveform, orig_freq=sample_rate, new_freq=16000)


        # Convert to mono
        if waveform.shape[0] > 1:
            waveform = torch.mean(waveform, dim=0, keepdim=True)

        # ASTFeatureExtractor looks for shape: (time, freq)
        inputs = self.feature_extractor(waveform.squeeze(0), sampling_rate=16000, return_tensors="pt")
        x = inputs["input_values"].squeeze(0)
        y = self.labels[idx]
        return x, y

def main(args):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    train_dataset = SpectrogramDataset(data_path=os.path.join(args.dataDir, "train"),
    labels_path=os.path.join(args.dataDir, "train.csv"))
    train_set, val_set, test_set = split_dataset(train_dataset)

    train_loader = DataLoader(train_set, batch_size=args.batch_size, shuffle=True)
    val_loader = DataLoader(val_set, batch_size=args.batch_size)
    test_loader = DataLoader(test_set, batch_size=args.batch_size)
    print("loaded")

    model = ASTForAudioClassification.from_pretrained("MIT/ast-finetuned-audioset-10-10-0.4593")
    print(model.classifier)

    model.classifier = nn.Linear(model.classifier.dense.in_features, 2)  # Binary classification
```

Results

Model	Input Size	Accuracy (%)	AUROC	AUPRC
CNN_108x108	108×108	93.2	0.945	0.921
InceptionV1_108x108	108×108	94.5	0.960	0.941
InceptionV1_75x45	75×45	94.3	0.958	0.937
EfficientNetV2-S	224×224 RGB	97.8	0.987	0.984
Audio Spectrogram Transformer (AST)	224×224 RGB	**98.1**	**0.991**	**0.989**

- All models trained for 10 epochs and consistent splits
  - AUROC: Area Under the Receiver Operating Characteristic curve
  - AUPRC: Area Under Precision Recall Curve
- Target AUROC >= 0.98 met by **AST & EfficientNet**
- **AST had highest Accuracy, AUROC, and AUPRC**
- EfficientNet 2nd best, and might be the better candidate in certain situations





Applications & Future Work

Applications

- Real-time whale detection on buoys to prevent ship strikes
- Automates labeling in massive underwater datasets
- Supports marine conservation and species monitoring
- Lightweight CNN model deployable on edge devices

Future Work

- Self-supervised pre training on unlabeled marine audio
- Model compression via knowledge distillation (AST -> lightweight student models)
- Extend classification to multiple species and ambient acoustic events
- Visualize AST attention maps for interpretability and trust

Conclusion

- Developed and benchmarked a deep learning pipeline for whale vocalization detection using spectrogram features.
  Evaluated four models including CNN, InceptionV1, EfficientNetV2-S, and AST.

- AST outperformed all models with **highest test accuracy (98.1%)**, AUROC (0.991), and AUPRC (0.989).

- EfficientNetV2-S was a close second, offering high performance with fewer parameters than AST.
- CenterLossSimple helped improve intra-class compactness, showing competitive accuracy and promising feature representations.

- The system demonstrates strong potential for real-time marine acoustic monitoring and can be adapted for multi-species classification in future research.