Dealing with Distribution Shift in Acoustic Mosquito Datasets

1st Hermann Y. Nkouanga
Computer Science Department
Portland State University
Portland, Oregon, 97201, USA
hermann@pdx.edu

2nd Suresh Singh
Computer Science Department
Portland State University
Portland, Oregon, 97201, USA
singh@cs.pdx.edu

***Abstract*—In recent years, the task of detecting mosquito presence through acoustic data has drawn the attention of many researchers. However, just like in any other detection task, these researchers are often confronted with the distribution shift problem, which alludes to the situation where the training and test datasets do not share the same distribution. A detection system is almost always guaranteed to fail during testing when this situation arises. Solutions to this issue have been proposed over the years, but they are often computationally expensive and complex to implement. In this paper, we propose a simple solution that consists in (1) identifying and getting rid of the noise present in the input data, (2) performing a dimensionality reduction, and (3) classifying the data. We tested our technique on a large and publicly available dataset of mosquito recordings (HumBugDB) and the results showed a maximum improvement of nearly 28% when compared to a baseline classification system.**

***Index Terms*—distribution shift, mosquito dataset, signal processing, convolutional neural network**

I. Introduction

Considered the deadliest animal on the planet, mosquitoes can carry and transmit life-threatening diseases such as malaria, yellow fever, dengue, and dog heartworm. However, these diseases, which affect millions of people worldwide every year [1], can be prevented by identifying places most susceptible to hosting dangerous species of mosquitoes and taking appropriate measures to prevent mosquito bites. With the recent increase in computing power and associated progress made in domains such as computer vision and machine learning, researchers developed solutions for automatically identifying such places as well as the species of mosquitoes they host.

A common issue with existing mosquito detection systems is that they need to be retrained every time the data distributions deviate from those of the set used to train the model. Possible causes for this distribution shift include changes of capturing device or environment, different background noise levels, and pollution of acoustic recording by other insect sounds. This issue has attracted the interest of many researchers over the last decade and numerous solutions leveraging frameworks such as domain adaptation, domain generalization, and sub-population shift have evolved. Even so, these solutions are often complex to implement and still do not really solve the performance drop issue [2]. Our goal when conducting this research was to come up with a new, simple, and efficient approach that mitigates the shortcomings of the existing ones. We focused on sound recording based classification of mosquitoes and experimented with the HumBugDB dataset [3] –a large, publicly available, and open-source dataset of mosquito acoustic recordings (see Section III-A).

The approach we used consisted in identifying and getting rid of Gaussian noise present in the input data, (2) performing a dimensionality reduction on the extracted features, and (3) classifying the data. This allowed us to improve the receiver operating characteristic curve (AUC) performance of our baseline classification model (BNN-Keras-4conv) by up to 28%.

This paper's contributions can be summarized as follows:

1) We provide a detailed review of existing work on the distribution shift.
2) We describe in detail a distribution shift testing technique and the results after applying it to HumbugDB.
3) We describe in detail an existing solution to distribution shift and the results after applying it to HumbugDB.
4) We present a new and simple solution to distribution shift and the results after applying it HumbugDB.

The remainder of this paper is structured as follows. Section II presents some related work. Section III goes over the solution we propose, its implementation, and our motivations. Section IV describes in detail the experiments that we conducted. Section V concludes the paper.

II. Related Work

In this section, we describe the distribution shift problem (II-A), two frameworks widely used to deal with it (II-B), and common classification schemes often used in acoustic-based classification and detection (II-C).

A. Problem Setting

According to Zhang et al. [4], three types of distribution shifts are often encountered in datasets, namely covariate shift, label shift, and concept shift.

Covariate shift happens when the distribution of a model's input data changes over time while the conditional distribution between labels and input data remains the same. Mathematically speaking, this means that the conditional distributions $P(y|x)$ ($x = input$ and $y = label$) for the source ($S =$

$training\ dataset$) and target ($T = testing\ dataset$) domains are identical, but the distributions for their input data are different.

Label shift is the inverse of covariate shift, i.e label shift happens when the distribution of labels changes but not the conditional distribution between inputs and labels.

Concept shift happens when the definition of labels changes across domains. This scenario may occur for example in a system that classifies individuals as healthy or non-healthy given that the criteria for being considered healthy may change over time.

B. Dealing with Distribution Shift

Two widely used techniques for correcting distribution are domain adaptation (DA) and domain generalization (DG)

DA is a form of transfer learning that tackles scenarios where a model is trained on a source distribution D_S then tested in the context of a different (but related) target distribution D_T. Its goal is to train a model f using samples from both D_S and D_T such that f makes as few mistakes as possible on D_T [5].

DG on the other hand deals with cases where the target data is inaccessible during training. Its goal is to train a model f' using samples from a source domain D_S such that f' can generalize well to an unseen and disjoint domain D_T, i.e $D_S \cap D_T = \emptyset$ [6].

C. Classification and Detection

Just like with image and text data, an audio signal needs to be pre-processed and transformed into a format acceptable by a classifier. Standard approaches for accomplishing this task include Fourier Transform and Wavelet Transform.

After being pre-processed, acoustic data is often fed to a classification model such as a convolutional neural network (CNN) or a convolutional recurrent neural network (CRNN). CNNs have been widely used for mosquito wingbeat detection and classification in recent years and the results are quite promising [7], [8], [9]. Popular CNN architectures often used in this domain include Resnet18, Resnet50, VGG13, VGG16, and DenseNet121.

III. Method

This section describes the dataset we used as well as the main steps we went through to classify audio recordings as mosquito sound or background noise.

A. Data Collection

For our experiments, we considered using the HumBugDB dataset [3]–a publicly available collection of audio recordings containing mosquito and background noise. We chose to work with it because the authors had already run some experiments on it and the results they reported showed some clear signs of bias. The dataset contains 9,295 audio files totaling approximately 35 hours of recording (20 hours of mosquito sounds from 36 species and 15 hours of background noise). The recordings were done under 8 different setups built from a set of different recording devices (low-cost smartphones and professional material), environments (wild and laboratory), and locations (Kenya, Tanzania, Thailand, the United Kingdom, and the United States of America). The recordings for six of them were combined and used as the training set (train) and those for the remaining two setups were held separately for testing purposes (test_A and test_B). We applied some pre-processing and data augmentation to these sets (see Section III-B). and ended up with 303,461 samples in train, 3782 samples in test_A, and 1445 samples in test_B.

B. Data Pre-processing

In this phase of the system, we borrowed the same pre-processing steps used by Kiskin et al [3]. Those steps can be summarized as follows:

- Load each input audio file.
- Split the audio input into 1.92 seconds audio chunks and assign them the same label.
- For each audio chunk in train, extract 128 log-mel spectrogram features with a time window of 30 feature frames and a stride of 5 frames. The resulting samples $X_i \in \mathbb{R}^{128\times30}$,
- For each audio chunk in test_A and test_B, extract 128 log-mel spectrogram features with a time window of 30 feature frames and a stride equal to the window's length. The resulting samples $X_j \in \mathbb{R}^{128\times30}$,
- Discard test audio chunks with a length shorter than 1.92 seconds.

C. Additional Data Processing Steps

To tackle the distribution shift issue observed in HumBugDB, we followed a procedure that can be classified under the domain generalization framework described in Section II-B. We tried to reduce the differences between samples from different domains (setups) in order to help the classification model learn domain invariant features. To do so, we proceeded as follows:

- We suppressed Gaussian noise from each sample to ensure that samples from all sets (train, test_A, and test_B) end up with about the same amount of noise.
- We reduced the dimensionality of the data using principal component analysis (PCA). The goal here was to get rid of non-essential and redundant information, which we believed could contribute to the disparity across the domains. It is worth mentioning that we also experimented with other dimensionality reduction techniques such as feature agglomeration (FA) and recursive feature elimination (RFE), but PCA appeared to be more efficient.

More details about our data processing steps are provided in Section IV-C where we discuss all our experiments.

D. Classification

In the classification stage of our system, we considered using the same binarized neural network (BNN-Keras-4conv) used by Kiskin et al. [3]. The configuration of the model remained the same except for the input layer which was

modified to match the size of input data after performing dimensionality reduction. We used this model because (1) it appeared to be efficient given the results reported for test_A [3] and (2) our focus was not on the network architecture but on how the data is processed. Refer to Section IV-B for more information about the model.

IV. Experiments

In this section, we describe the metric we considered to evaluate the outcome of our experiments (IV-A), the classifiers that we used (IV-B), and the experiments themselves with corresponding results (IV-C).

A. Evaluation Metric

As evaluation metric, we considered the area under the receiver operating characteristic curve (AUC). We followed a ten Monte Carlo dropout samples protocol as used by Kiskin et al. [3]. Therefore, the results reported in Section IV-C are the averages of the ten samples.

B. Classifier

In the classification stage of our experiments, we considered the three models used by Kiskin et al. [3], i.e BNN-Keras-4conv, BNN-ResNet-50, and BNN-ResNet-18.

- BNN-Keras-4conv is a CNN implemented using Keras [11]. It contains four convolutional layers, two pooling layers (one after each of the first two convolutional layers), two fully-connected layers, and five lambda layers (one after each pooling layer, one after each of the last two convolutional layers, and one between the two fully-connected layers).
- For BNN-ResNet-50 and BNN-ResNet-18, we used their corresponding implementations available in the PyTorch library [12]. For both implementations, the fully connected layer was modified to have one neuron (instead of 1000) for binary classification.

C. Results

Experiment 1: In our first experiment, we tested HumBugDB for distribution shift. To accomplish this, we used a technique called subsample shift test proposed by Betthauser et al. [13]. The test procedure between two datasets X and Y is as follows (See Betthauser et al. [13] for the complete algorithm):

1: Select a distance metric D, a subsample size m, and a number of subsamples n
2: Generate two subsamples/subsets x_1 and x_2 of size m from X
3: Compute $D_{xx} = D(x_1, x_2)$
4: Generate two subsamples x and y of size m from X and Y respectively
5: Compute $D_{xy} = D(x, y)$
6: Compare D_{xx} and D_{xy}
7: Repeat steps 2-6 n times
8: Draw a conclusion

TABLE I: AUCs obtained when swapping some percentage of test_A samples and test_B samples with the same corresponding number of samples from train.

Percentage of samples	test_A (AUC) R-50	test_B (AUC) R-50
10%	0.99	0.90
20%	0.99	0.96

Before applying the steps listed above, we reduced the dimensionality of train, test_A, and test_B from 3840 to 512 features using PCA for faster processing. We used $m = 1024$ and $n = 50$. For the distance metric, we used energy distance (ED) and Wasserstein distance (WD), which are well-known metrics for measuring the distribution distance between random vectors. We tested train vs test_A, train vs test_B, and test_A vs test_B separately and the results can be seen in Figures 2, 3, and 4 respectively.

As Figures 2, 3, and 4 show, both ED and WD are considerably low when subsamples are generated from the same dataset (i.e blue dots) compared to when they originate from different datasets (i.e pink dots). Based on this large differences, we can conclude that indeed there is some sort of distribution shift across train and test_A, train and test_B, and test_A and test_B.

Experiment 2: In our second experiment, our goal was to figure out why all the classification algorithms (see BNN-Keras-4conv, BNN-ResNet-50, and BNN-ResNet-18) perform well on test_A but poorly on test_B as reported by Kiskin et al. [3]. Our intuition was that test_A's domain was closer to train's domain than test_B's domain. To verify our hypothesis, we proceeded by combining train and test_A and clustering the resulting set into seven clusters using k-means [14]. We did the same thing for train and test_B. We wanted to observe if a clustering algorithm could distinguish test_B from train better than test_A. As expected, 84% of test_B samples were recognized as belonging to the same cluster compared to only 74% for test_A. After confirming our hypothesis, we wanted to test if incorporating some samples from test_B into train would yield better results.

Experiment 3: In this experiment, we withdrew 10% of samples from each test dataset and swapped it with the same number of samples withdrawn from train. We then trained the BNN-ResNet-50 model using the resulting train dataset and tested it on the resulting test_A and test_B datasets. We repeated the same experiment but we withdrew and swapped 20% of samples this time. The results for both trials, as shown in Table I, show that the more samples from test distributions are represented in the training dataset, the better the model performs.

Experiment 4: In our first attempt to reduce the effect of distribution shift on our test sets, we considered a well known technique called weighted empirical risk minimization. We followed the implementation proposed by Zhang et al. [4] which includes the following steps:

- Combine all three datasets, i.e train, test_A, and test_B,

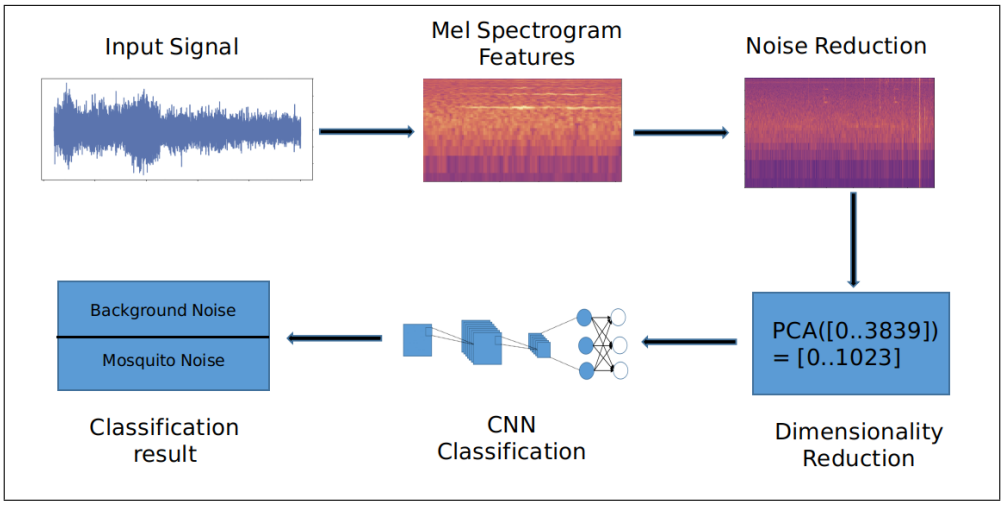

Fig. 1: System architecture. Our focus for this research is on phases 3 (Noise Reduction) and 4 (Dimensionality Reduction).

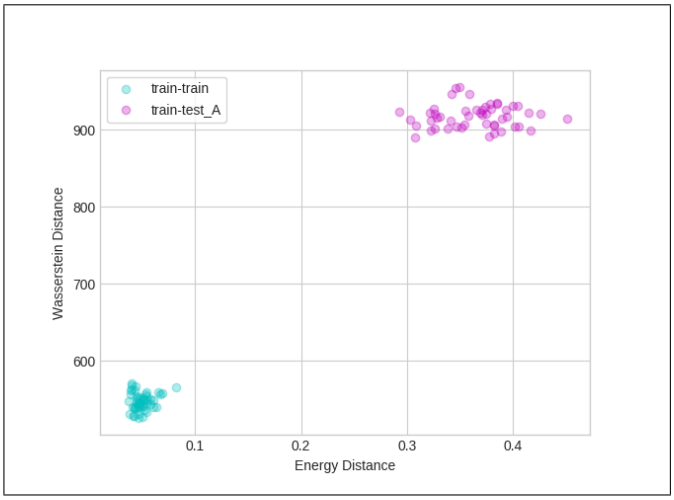

Fig. 2: ED vs WD (train vs test_A).

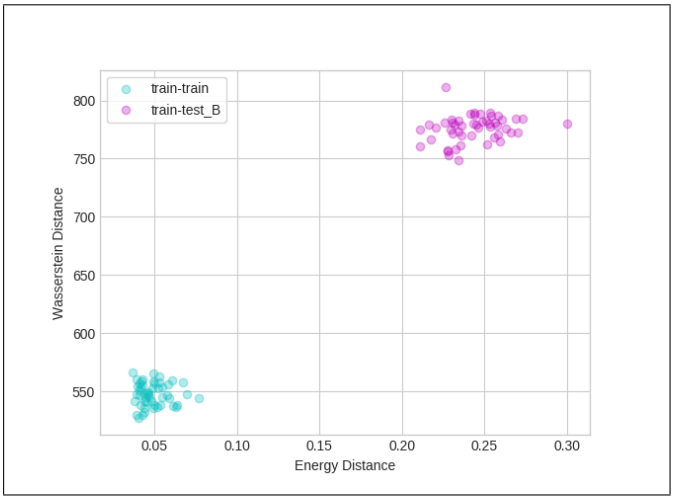

Fig. 3: ED vs WD (train vs test_B).

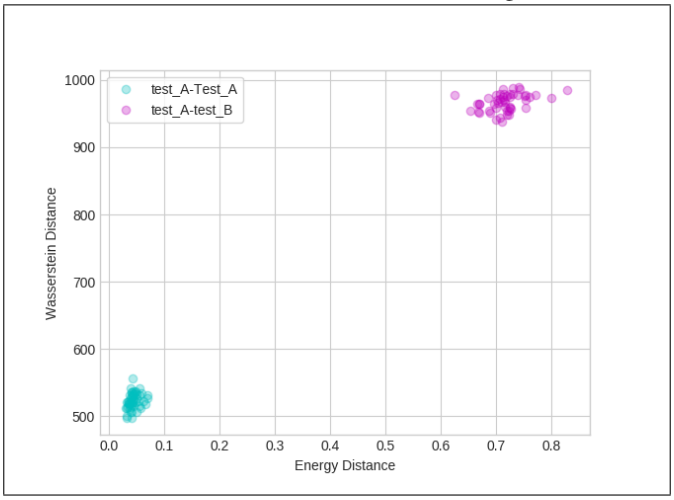

Fig. 4: ED vs WD (test_A vs test_B).

and assign labels 1, 2, and 3 respectively to their corresponding samples.

- Train a logistic regression model with this resulting dataset to obtain a probability estimator function h. For

TABLE II: Summary of results obtained after applying the weighted empirical risk minimization approach (WERM). K-BNN ≡ BNN-Keras-4conv, R-50 ≡ BNN-ResNet-50, and R-18 ≡ BNN-ResNet-18.

Operation	test_A (AUC) K-BNN	test_B (AUC) K-BNN	test_A (AUC) R-50	test_B (AUC) R-50	test_A (AUC) R-18	test_B (AUC) R-18
WERM	0.88	0.35	0.90	0.55	0.94	0.55

this model, we used the implementation provided by the SciKit-Learn python library [10] along with the default parameter, i.e random_state = 0.

- Use function h to compute the probabilities for each sample in train to belong to train, then compute the exponential for each of the resulting values.
- finally, train the three models (BNN-Keras-4conv, BNN-ResNet-50, and BNN-ResNet-18) using train and the resulting numbers from the previous step as sample weights.

We tested the models on test_A and test_B and obtained the results reported in Table II. These results show that the models are still performing a lot better on test_A than on test_B, i.e the weighted empirical risk minimization approach is not very efficient for test_B.

Experiment 5: In this experiment, we tried to find the source of the distribution shift across our three datasets. Our first intuition was that the three sets contain different levels of noise. To verify this hypothesis, we tried adding and/or suppressing noise to/from one or more of the datasets, then retrained (when necessary) and tested the classifier to see if its performance would improve. To add noise to the datasets, we used NumPy's function numpy.random.normal [16] and experimented with different values for the mean and standard deviation (Std) (see Table III). To suppress noise, we used SciPy's savgol_filter [17] function with parameter window_length=121. It is worth noting here that we experimented with different values for the window's length and 121 is what provided the best results. We recorded in Table III the most interesting results along with implementation details for the trials we made. These results show that adding Gaussian noise with mean = 1.0 and Std = 0.4 produced the best overall result (0.94 + 0.62 = 1.56). However, we can see that the difference between test_A and test_B AUCs (0.32) is still significant i.e that adding noise may improve the results but does not really solve the distribution shift issue. It is worth mentioning that our baseline classification results were obtained using the code made available by Kiskin et al. [3] on their GitHub page [15]. However, we unfortunately were not able to reproduce the same results as reported by the authors (See first two rows of Table III)

Our second intuition was that the high dimensionality of the data (3840 features per sample) could cause some features to be redundant and/or irrelevant. The number of such features could vary very slightly across samples within the same domain but considerably across samples from different domains.

TABLE III: Summary of results obtained after adding or removing noise to/from the entire dataset (i.e train, test_A, and test_B). Baseline classification ≡ classification of the extracted log-mel spectrogram features without applying any further transformations to the data. The numbers in bold represent the highest AUC in their respective column.

Operation	mean	std	test_A (AUC) K-BNN	test_B (AUC) K-BNN	test_A (AUC) R-50	test_B (AUC) R-50	test_A (AUC) R-18	test_B (AUC) R-18
Kiskin et al. [3] (Humbug Paper)	-	-	**0.96**	0.35	**0.96**	0.55	0.92	**0.67**
Baseline Classification	-	-	0.81	0.41	0.94	0.49	0.78	0.54
Adding Noise	0.5	0.4	0.85	0.37	0.88	0.62	0.92	0.54
Adding Noise	1.0	0.4	0.74	0.42	0.82	**0.67**	0.94	0.62
Adding Noise	1.5	0.4	0.66	0.48	0.79	0.64	**0.95**	0.51
Adding Noise	2	0.2	0.80	0.47	0.82	0.64	0.93	0.47
Reducing Noise	-	-	0.58	**0.61**	0.51	0.32	0.43	0.37

TABLE IV: Summary of results obtained after reducing the number of features for each sample using different dimensionality reduction techniques.

Operation	# Features	test_A (AUC) K-BNN	test_B (AUC) K-BNN	test_A (AUC) R-50	test_B (AUC) R-50	test_A (AUC) R-18	test_B (AUC) R-18
FA	1920	**0.81**	0.36	**0.74**	0.47	**0.62**	0.59
PCA	1024	0.72	**0.78**	0.37	0.75	0.36	0.75
PCA	512	0.55	0.77	0.36	**0.77**	0.36	**0.78**

Consequently, we thought that getting rid of such features could help our classifier to perform better. To test this hypothesis, we applied different dimensionality reduction algorithms to all three sets, followed by the train and test procedures. The dimensionality reduction techniques we experimented with include principal component analysis (PCA), feature agglomeration, and recursive feature elimination (RFE). The most interesting results and details for the trials we made are recorded in Table IV. These results show that test_A's AUC tends to get worse while test_B's AUC gets better as the number of features decreases. This can be explained by the fact that test_A initially has fewer redundant features compared to test_B (hence, the good baseline classification AUC for test_A), therefore important features start getting eliminated from test_A in order to reach the aimed dimensionality. Nevertheless, the results also show that the PCA technique works best overall (0.72 + 0.78 = 1.5) using the BNN-Keras-4conv classifier and 1024 as the number of features. We can also note an increase of test_B's AUC by 0.37 for only a decrease of test_A's AUC by 0.09 when compared to the baseline classification results in Table III for BNN-Keras-4conv.

Our third intuition was that the distribution shift could be due to both the redundant features and the difference in noise levels. Therefore, we tried to combine both adding/reducing noise and applying feature reduction to all three datasets. Since PCA provided the most promising results (see Table IV), we decided to only experiment with it. For adding noise, we created Gaussian noise with $mean = 0.5$ and $Std = 0.4$ and added it to the three datasets. For denoising, we kept the window's length at 121 as in the previous experiments. After applying these transformations to the datasets, we retrained

TABLE V: Summary of results obtained after combining noise addition/reduction and dimensionality reduction using PCA. These results were obtained using mean = 0.5 and Std = 0.4 for noise addition and window_length=121 for noise reduction.

Operation	# Features	test_A (AUC) K-BNN	test_B (AUC) K-BNN	test_A (AUC) R-50	test_B (AUC) R-50	test_A (AUC) R-18	test_B (AUC) R-18
Adding Noise + PCA	1024	0.65	0.75	0.37	0.70	0.37	**0.78**
Adding Noise + PCA	512	0.65	**0.78**	0.38	**0.74**	0.34	0.73
Reducing Noise + PCA	1024	0.82	0.69	0.64	0.63	0.63	0.63
Reducing Noise + PCA	512	**0.83**	0.65	**0.70**	0.68	**0.68**	0.66

and tested the model and the results that we obtained are recorded in Table V. These results show that reducing noise and applying PCA (with number of features = 1024) produced the best overall AUC (0.82 + 0.69 = 1.51). This means that we were able to increase test_A's AUC by 0.01 and test_B's AUC by 0.28 when compared to the baseline classification results in Table III for the BNN-Keras-4conv classifier.

To summarize this section our results show that we were able to obtain maximum AUCs of 0.95 and 0.78 for test_A (see Table I) and test_B (see Tables IV and V) respectively. These results show an increase of 0.11 AUC for test_B for a decrease of only 0.01 AUC when compared to the maximum AUCs reported by Kiskin et al. [3] (see Table III). Nevertheless, none of the techniques we used produced, alone, acceptable results for both testing datasets. Even so, the approach of reducing noise and data dimensions to 1024 using PCA along with the BNN-Keras-4conv classifier produced the most promising results when comparing the results for all the approaches we explored to those of their corresponding baseline classification (see Table III).

V. Conclusions

In this paper, we explored different simple approaches for dealing with the distribution shift issue often found in machine learning datasets. We focused on mosquito sound recording datasets in general and more specifically on the Humbug dataset [3], which contains three sub-datasets (train, test_A, and test_B) collected in different environments and/or recording setups. We attempted different approaches including adding or suppressing Gaussian noise from the dataset, dimensionality reduction, and a combination of noise suppression and dimensionality reduction. The latter approach appeared to be the most promising when tested on a convolutional neural network implemented using the Python machine learning library Keras [11]. It yielded 0.82 AUC for test_A and 0.69 AUC for test_B, which correspond to increases of 0.1 and 0.28 respectively when compared to a baseline classification using the same model.

Even though we consider our results to not be sufficient, we believe that our research opens doors to other potential lines of work. For example, it would be interesting to experiment with other audio processing techniques such as the wavelet transform, which when compared to the Fourier transform used in this paper, is theoretically guaranteed to provide more meaningful features. It would also be interesting to experiment with other model architectures, denoising algorithms, and dimensionality reduction techniques.

References

[1] W. H. Organization, "Malaria," https://www.who.int/news-room/fact-sheets/detail/malaria, DEC 2021, accessed on 2021-15-12.

[2] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao, T. Lee, E. David, I. Stavness, W. Guo, B. Earnshaw, I. Haque, S. M. Beery, J. Leskovec, A. Kundaje, E. Pierson, S. Levine, C. Finn, and P. Liang, "Wilds: A benchmark of in-the-wild distribution shifts," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 5637–5664. [Online]. Available: https://proceedings.mlr.press/v139/koh21a.html

[3] I. Kiskin, M. Sinka, A. D. Cobb, W. Rafique, L. Wang, D. Zilli, B. Gutteridge, R. Dam, T. Marinos, Y. Li, D. Msaky, E. Kaindoa, G. Killeen, E. Herreros-Moya, K. J. Willis, and S. J. Roberts, "Humbugdb: A large-scale acoustic mosquito dataset," *CoRR*, vol. abs/2110.07607, 2021. [Online]. Available: https://arxiv.org/abs/2110.07607

[4] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2020, https://d2l.ai.

[5] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. S. Lempitsky, "Domain-adversarial training of neural networks," in *Domain Adaptation in Computer Vision Applications*, ser. Advances in Computer Vision and Pattern Recognition, G. Csurka, Ed. Springer, 2017, pp. 189–209. [Online]. Available: https://doi.org/10.1007/978-3-319-58347-1_10

[6] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *CoRR*, vol. abs/2103.02503, 2021. [Online]. Available: https://arxiv.org/abs/2103.02503

[7] Ivan Kiskin, Bernardo Pérez Orozco, Theo Windebank, Davide Zilli, Marianne Sinka, Kathy Willis, and Stephen J. Roberts, "Mosquito detection with neural networks: The buzz of deep learning." *CoRR*, 2017.

[8] M. S. Fernandes, W. Cordeiro, and M. Recamonde-Mendoza, "Detecting aedes aegypti mosquitoes through audio classification with convolutional neural networks," *Computers in Biology and Medicine*, vol. 129, p. 104152, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482520304832

[9] E. Fanioudakis, M. Geismar, and I. Potamitis, "Mosquito wingbeat analysis and classification using deep learning," in *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 2410–2414.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] TensorFlow, "Convolutional neural network (cnn)," https://www.tensorflow.org/tutorials/images/cnn, accessed on 2021-28-12.

[12] PyTorch, "Resnet," https://pytorch.org/hub/pytorch_vision_resnet/, accessed on 2021-28-12.

[13] L. Betthauser, U. Chajewska, M. Diesendruck, and R. Pesala, "Discovering Distribution Shifts using Latent Space Representations," *arXiv e-prints*, p. arXiv:2202.02339, Feb. 2022.

[14] Scikit-Learn, "sklearn.cluster.kmeans," https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, accessed on 2021-28-12.

[15] I. Kiskin, "Humbugdb," https://github.com/HumBug-Mosquito/HumBugDB, accessed on 2021-21-8.

[16] T. N. community, "numpy.random.normal," https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html, accessed on 2021-28-12.

[17] T. S. community, "scipy.signal.savgol_filter," https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html, accessed on 2021-28-12.

[18] Scikit-Learn, "Dimensionality reduction," https://scikit-learn.org/stable/index.html, accessed on 2021-28-12.