University of Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia

Deep learning techniques for biological signal processing: Automatic detection of dolphin sounds

Supervisor
Dr. Alberto Testolin
University of Padova

Co-supervisor
Dr. Roee Diamant
University of Haifa

Master Candidate
Burla Nur Korkmaz

Student ID
2008484

Academic Year
2021-2022

Abstract

Considering the heterogeneous underwater acoustic transmission context, detecting and distinguishing vocalizations of cetaceans has been a challenging area of recent interest. A promising venue to improve current detection systems is constituted by machine learning algorithms. In particular, Convolutional Neural Networks (CNNs) are considered one of the most promising deep learning techniques, since they have already excelled in problems involving the automatic processing of biological sounds. Human-annotated spectrograms can be used to teach CNNs how to distinguish between information in the time-frequency domain, thus enabling the detection and classification of marine mammal sounds. However, despite these promising capabilities machine learning suffers from a lack of labeled data, which calls for the adoption of transfer learning to create accurate models even when the availability of human taggers is limited. In this thesis, we developed a dolphin whistle detection framework based on deep learning models. In particular, we investigated the performance of large-scale pre-trained models (VGG16) and compared it with the performance of a vanilla Convolutional Neural Network and several baselines (logistic regression and Support Vector Machines). The pre-trained VGG16 model achieved the best detection performance, with an accuracy of 98,9% on a left-out test dataset.

Contents

Listing of figures

Listing of tables

1 Introduction

Analyzing and detecting dolphin vocalizations has piqued attention in recent years, posing a challenge to maritime mammalogists and other scientists [1, 2, 3]. Whistle, squawk, chirp, moan, burst pulses, and clicks are examples of vocalizations [4]. Whistles are the dolphins' primary form of communication, and they play an important role in their daily lives [5, 6]. Researchers hope to protect their populations and better understand their behaviors by analyzing their vocalizations, and whistles [7, 8, 9]. Whistles are made in groups for a variety of reasons, including play behaviors, conspecific rubbing and hostility [10], reunions of mother and her calves [11], and identification [12]. Whistles have also been observed regularly during their social interactions by mammalogists [13], exposing a communication purpose. Dolphin whistles are frequently recorded below 20 kHz [14].

Researchers can obtain spectrograms from underwater sounds and manually detect and identify the dolphin whistles. While this strategy has certain benefits, it also has some disadvantages. This method's main merits are its simplicity and convenience of use, but it also has some major drawbacks. Due to its reliance on the operator's physical and psychological circumstances, this method can be unreliable and, extremely, requires a high amount of labor and attention. Besides this method, artificial intelligence and machine learning based frameworks can be used to detect and identify spectrograms automatically more efficiently and with higher performance [15]. In terms of the amount of data to be inspected on a time basis, artificial intelligence will be considerably faster than humans.

Artificial Intelligence (AI) is described as a program with cognitive abilities comparable to

those of a person. One of the major concepts of artificial intelligence is to have computers think like people and solve problems in the same manner we do. However, in addition to their incredible successes, AI systems implemented as artificial neural networks have faced a sizeable number of difficulties. Since the data needed to be handled in mammal species sound analysis is very large, automation of the detection process has high importance for efficiency. The research work described in this thesis was carried out under the supervision of Professor Alberto Testolin from the University of Padova. It aims to create an accurate and automatic dolphin whistle detection pipeline to be used in marine mammal species sound analysis.

In our work, we used machine learning and deep learning algorithms to evaluate and compare the efficiency of automatic detection methods with the spectrograms we created considering the 3 kHz – 20 kHz interval. Pre-trained VGG16 [16], a vanilla Convolutional Neural Network (CNN) [17], and machine learning algorithms Logistic regression and Support Vector Machines (SVMs) were trained and compared. Data augmentation and cross-validation techniques were also applied to see how much they affected the results. Here are our main contributions in this work as follows:

- An accurate dolphin whistle detection from underwater audio recordings.
- Comprehensive comparison of state-of-the-art machine learning and deep learning methods.
- Comparison of the effects of data augmentation, transfer learning and cross-validation in the training phase.

The structure of our work is as follows. chapter 2 discusses the related works involved in the machine learning algorithms for marine mammal sound detection. chapter 3 describe commonly used machine learning and deep learning algorithms. chapter 4 represents the details of the technologies used in the implementation of our project. chapter 5 represents the general explanation of our problem and the details of the signals, the dataset we created, and the data preprocessing techniques we used with the evaluation metrics. chapter 6 and chapter 7, respectively, discuss models and evaluations of their performance. Lastly, chapter 8 brings the report to a close with additional interpretations and suggestions for further research.

2
Related Work

Detecting dolphin sounds and making analyses on them has been a very interesting research area lately, and various approaches have been developed. In this section, we will focus on different approaches to dolphin sound detection. For example, in this work [18], the ridge detection algorithm method was used on the obtained spectrograms. In this way, they acquired areas of the image that have significant dark patches. Later on, the Hough transform [19] was applied to the binarized maps obtained by the Ridge algorithm [20] for detecting line forms. The active contour algorithm has been applied to detect the shapes to seize the shape of these patterns in addition to the Hough transform. Finally, the Random Forest classifier [21] was applied, and 97.7% accuracy was obtained. This study [22] is based on CNN and they used LeNet, LeNet variants, and ResNet-18 [23] architectures to detect dolphin echolocation clicks. Among their experiments, ResNet-18 gave the best result with 97.44% accuracy. Also, in this work [24], they are utilizing different types of deep neural network architectures, for example, convolutional neural networks (CNN) and recurrent neural networks (RNN) [25] to detect the vocalizations of North Atlantic right whales. They also implemented hard negative mining and data augmentation to improve deep neural network results. Here [3], they are proposing using pre-trained AlexNet to detect and classify marine mammals using colored spectrograms using the Watkins database. The detection and classification models have an accuracy of 99.96% and 97.42%, respectively. In [26], different ResNet architectures are trained with and without max-pooling in the first residual layer to detect killer whale sounds, and the results are compared. Accuracies are distributed between 92% and 97%. In [27], a semantic segmentation approach is applied

by manually masking and labeling the dolphin whistles. DeepLabv3+, which uses a structure of encoder-decoder, a CNN model, is used. With an overall mean precision of 0.96, an accuracy of 0.89, and an F-score of 0.86, their semantic segmentation approach detects the whistle. Here [28], the authors aimed to remove the noise from the background by applying denoising on the raw sound in their work. All denoised sounds are converted into spectrograms without overlapping between adjacent frames. LeNet5 architecture is used for both detection and classification. Results reveal that the suggested strategy can get a 97% accurate detection rate and a 95% correct classification rate. In this paper [29], Mel Frequency Cepstral Coefficients (MFCCs) and Discrete Wavelet Transforms (DWTs) are applied for feature extraction from North Atlantic Right Whale up-calls. Later on, SVM and KNNs were used. It was observed that DWTs used in combination with MFCCs improved the results, and the up-call detection rate was 92.27%. In this study [1], a dolphin whistle detection technique was created, and a multi-layer perceptron neural network was suggested (MLPNN). The MLPNN parameters are initially optimized using the chimp optimization method. The authors [2] suggested an autoencoder built from convolutional and recurrent layers for usage with short windows of acoustic dolphin sounds. Encoder-decoder models used in natural language processing served as inspiration for their architecture. The authors of this article [30] describe a learning-based technique for obtaining toothed whale whistles. To learn to anticipate whistle contours, their method is done by using a small number of human and machine-generated annotations, which are whistle contours or edges in image spectrograms. Each hidden layer in their network, which consists of 10 convolutional layers, has 32 filters, and there are batch normalization layers behind all convolutional layers on residual blocks.

3

State-of-the-art

3.1 Machine learning Algorithms

3.1.1 Support Vector Machines

SVMs are linear models mostly applied to classification and regression problems. However, they can be used on data that cannot be separated linearly by applying different kernels. If we give an example over a linearly separable dataset on a two-dimensional plane, as can be seen from Figure 3.1, the line we call hyperplane divides the data into two different classes. As can be seen in this scenario, there is more than one hyperplane that can separate the data. SVM calculates the distance between the line and the support vectors in order to find the line that separates the data in the best and most optimal way. The distance between the support vectors and the hyperplane is called the margin. The hyperplane with the highest margin is considered the most optimal hyperplane.

In reality, datasets are much more complex and often cannot be separated linearly. As a solution to this situation, SVM has two solutions, and these are the Soft Margin and Kernel Tricks methods.

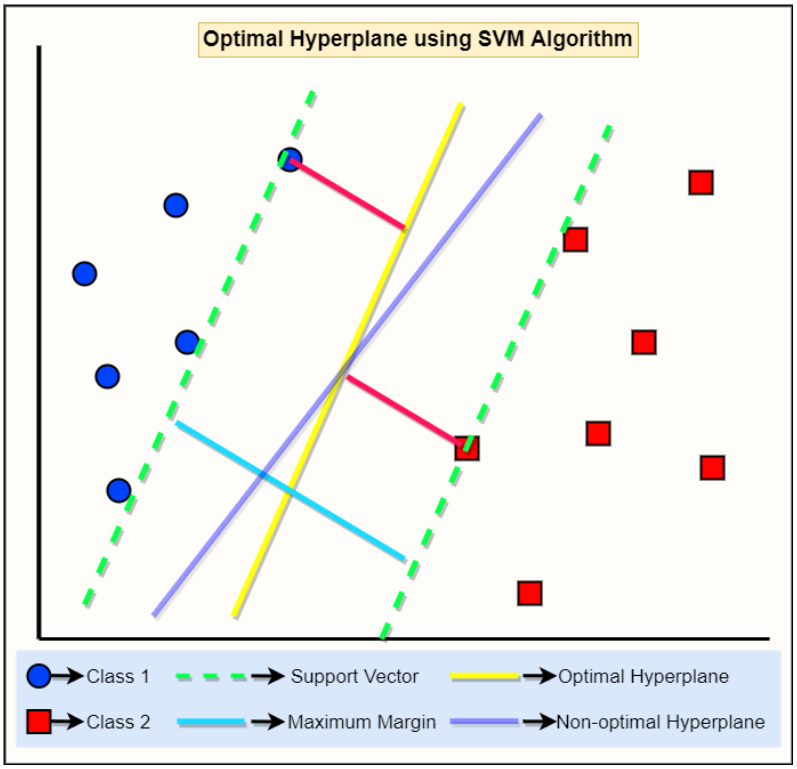

Figure 3.1: Support vector machines visualization

Soft Margin: The soft margin method means tolerating a few misclassified samples while trying to locate the hyperplane to separate the dataset. How much tolerance we apply is an important hyperparameter for SVM. In the Sklearn library used with the Python programming language, this parameter is represented as a penalty term and is symbolized as "C". A high C value indicates you'll get more training points right, which means the penalty when SVM makes a mistake will be higher, and SVM will generate more complicated decision boundaries in order to fit all the points.

Kernel Trick: Applying a kernel function to inputs in the original lower-dimensional space is known as a kernel trick. The inner product of the transformed vectors in the higher-dimensional space is returned by this function as shown in Figure 3.2. If we show it in a formal way; let's say our data $x, z \in X$, and a map $\varphi : X \to R^N$, then the formula is represented Equation 3.1.

$$k(x, z) = \langle \varphi(x), \varphi(z) \rangle \tag{3.1}$$

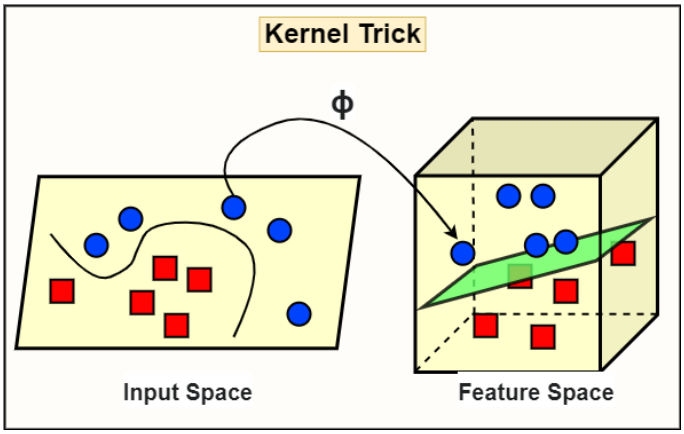

Figure 3.2: Kernel trick visualization

Polynomial Kernel: The polynomial kernel is a transformer that generates new features by combining all existing features in a polynomial way. For example, let's say that the feature vector we have is $X = [-2, -1, 0, 1, 2]$ with their labels $[1, 1, 0, 1, 1]$. As can be seen in Figure 3.3, it represents a dataset that cannot be separated linearly. Let's square the vector X, which is X^2. This time, the new vector we have will be $[4, 1, 0, 1, 4]$. In this way, the polynomial kernel can create a nonlinear decision boundary and separate the data.

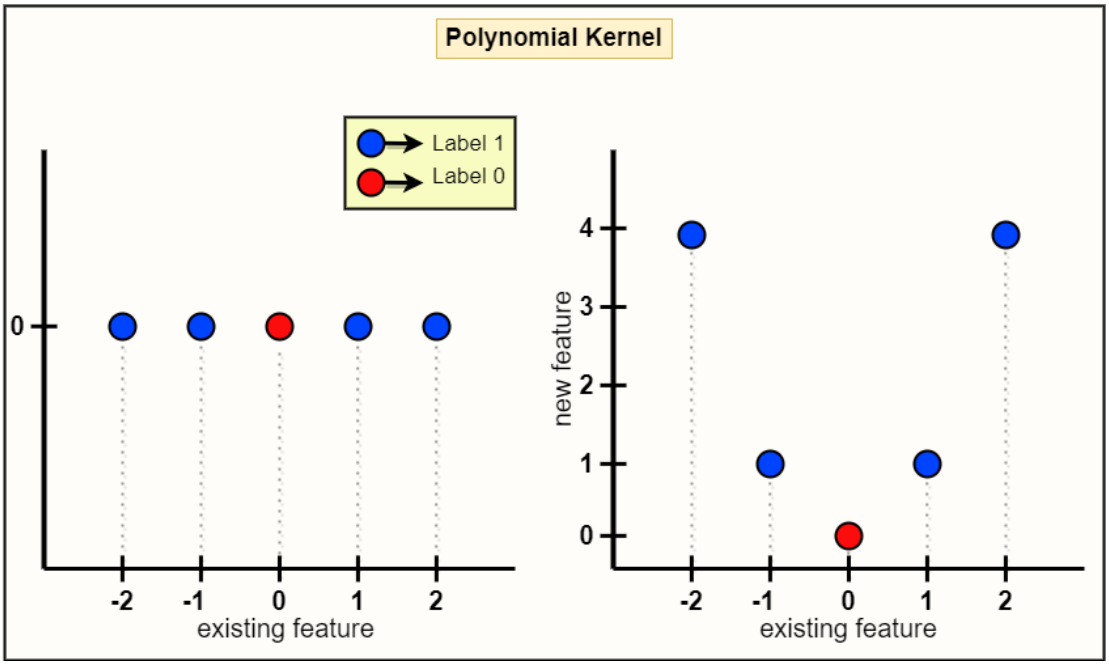

Figure 3.3: Example of polynomial kernel implementation

Radial Basis Function (RBF) kernel: The radial basis function is a kernel that measures the distance between all other data points and a specific data point which is the center to generate new features. Generally, the most used radial basis function is the Gaussian Radial Basis Function which is represented as Equation 3.2.

$$\varphi(x, center) = exp(-\gamma||x - center||^2) \tag{3.2}$$

γ is the gamma value that determines how new features affect the decision boundary. The higher the gamma, the greater the impact of the features on the decision boundary. If we go over the same examples again, let $X = [-2, -1, 0, 1, 2]$ and $label = [1, 1, 0, 1, 1]$. Let's choose the point (-1.0) as the center point and get the gamma value of 0.1. When we put the values into the Gaussian RBF kernel, the results will be as seen in Figure 3.4.

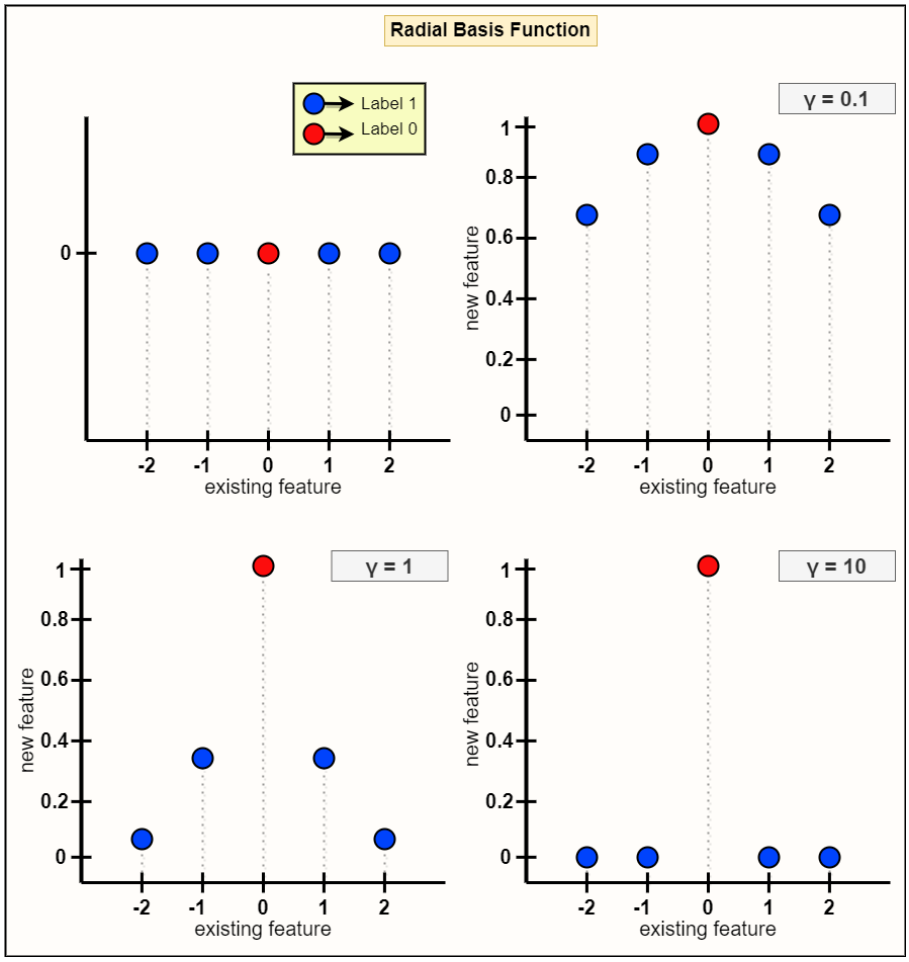

Figure 3.4: Example of radial basis function implementation

3.1.2 Logistic Regression

Logistic regression produces a dependent variable output by using independent variables as in linear regression, this output is our prediction result. Unlike linear regression, while the inputs can be categorical and numeric, the output is always categorical such as yes or no, passed or failed, etc., since it is a binary classification problem.

Given that the output of logistic regression is a probability between 0 and 1, the explication of the weights or coefficients is different from that of the weights in linear regression. The likelihood is not linearly governed by the weights anymore. The logistic function converts the weighted sum into a probability. The logistic regression model uses the sigmoid function Equation 3.3 as it is defined below to constrain the outcome of a linear equation between 0 and 1

instead of fitting a line or a hyperplane.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

3.2 Deep Neural Networks

Deep neural networks are models derived from artificial neural networks. Each node in the hidden layer of artificial neural networks plays a role in determining the importance of each input to define the output. In order to benefit more from this situation, more than one hidden layer has been used, and the increase in the number of hidden layers has revealed the concept of deep neural networks, as can be seen in Figure 3.5.

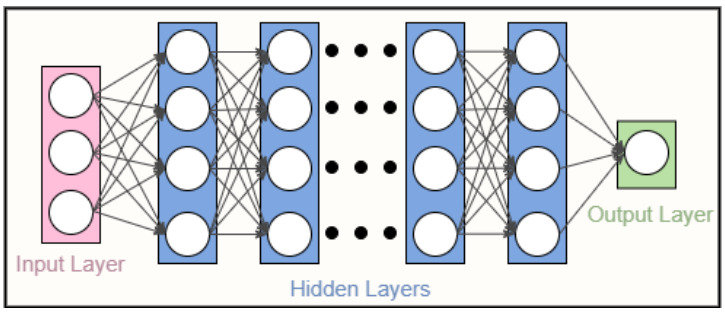

Figure 3.5: Deep neural networks visualization

Deep neural networks bring with them the importance of hyperparameter tuning. The parameters that vary depending on the problem and data set are called hyperparameters, and they are left to the individual who built the model. Hyperparameters are used when learning a neural network, but they are not part of the final model, like weights/coefficients and biases. Here are some typical hyperparameter examples:

- The rate of separation of the dataset as a train and test,
- Learning rate for optimization algorithms such as gradient descent,
- Which optimization algorithm to choose such as gradient descent, stochastic gradient descent, AdaGrad, or Adam,
- Which cost or loss function that the model will employ,

- Number of hidden layers,
- Number of nodes/units in hidden layers,
- Dropout rate,
- Number of epochs which is the number of iterations in training,
- Kernel or filter size in convolutional layers,
- Batch size,
- Pooling size,
- Stride size in convolution layers and pooling layers.

3.2.1 Fully Connected Neural Network

Fully Connected Neural Networks mean that each node of each layer is connected to the other. In Figure 3.6, We can see the reason why layers are referred to as Fully Connected in some cases. They can be called Fully Connected layers or Dense layers. Each input of the input vector affects each output of the output vector since they are all connected to each other. On the other hand, not all weights have an impact on all outputs. The connections of the first neuron in the layer are shown by the red lines. This neuron's weights just have an impact on output A; they have no bearing on outputs B or C.

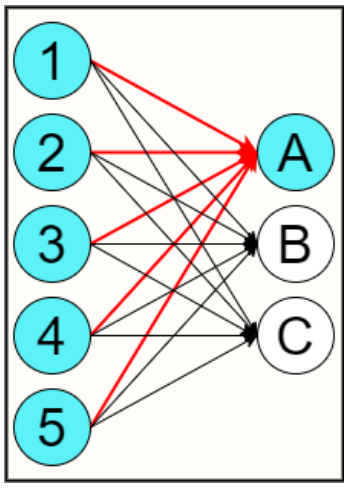

Figure 3.6: Fully connection explanation

A group of dependent non-linear functions makes up neural networks. A neuron is the building block of each distinct function. The neurons inside fully connected neural networks process the input vector linearly using a weights matrix. The output is then subjected to a non-linear transformation by applying a non-linear activation function f. Where W represents the weights matrix and x is the input vector, a dot product between them happens. w_0 represents the bias that can be included in activation function f can be seen in Equation 3.4.

$$y_{jk}(x) = f\left(\sum_{i=1}^{n_H} w_{jk} + w_{j0}\right) \tag{3.4}$$

A Fully Connected Neural Network layer with input size of 5 and output size of 3 can be visualized as shown in Figure 3.7 as an example. The activation function f receives as input the dot product between the layer's input and the weights matrix of this layer and returns us the output. These weights will be learned by the model with the help of optimization algorithms such as gradient descent during the training of the model. The weights matrix is a $5x3$ matrix, while the input is a $1x5$ vector. We obtain the $1x3$ output vector by using the dot product and a non-linear transformation using an activation function f.

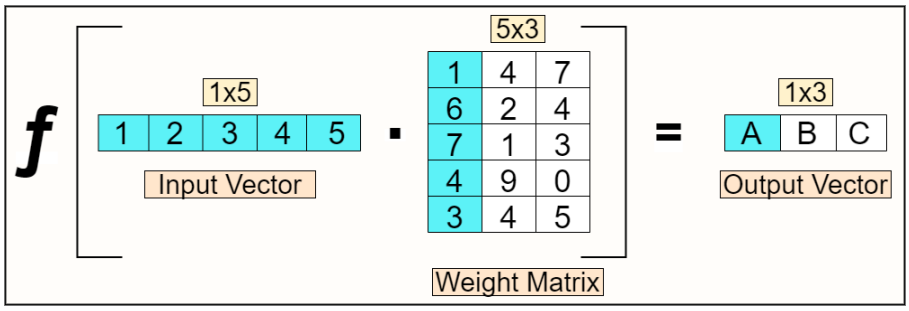

Figure 3.7: Weighted sum representation

3.2.2 Convolutional Neural Network

A convolutional neural network is a subclass of neural networks that are particularly adept at handling input with a grid-like architecture, like an image. A convolution, like a typical neural network, is a linear operation that includes multiplying a set of weights with the input. The multiplication is done between an array of input and a two-dimensional array of weights, called a filter or a kernel, as seen in Figure 3.8, because of approach was created for two-dimensional input. Despite the fact that CNN is created for two two-dimensional operations, it may also be utilized to process one and three-dimensional data. CNN employs technology similar to a

multilayer perceptron that is optimized for limited computational requirements. A layer for input, a layer for output, and a hidden layer, together with several convolutional layers, pooling layers, fully connected layers, and normalization layers, make up a CNN's layers.

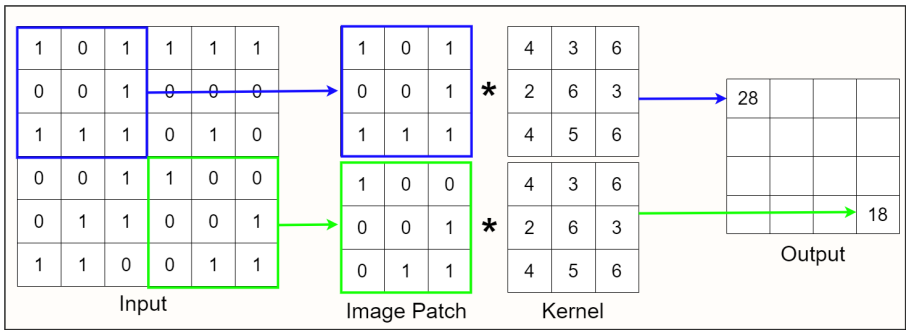

Figure 3.8: Convolution representation

Convolution is a widely used method in the field of computer vision, and different filters have been designed to facilitate the analysis of images. By applying these filters to the image, a feature map is obtained from it. The examples for these filters can be seen at Figure 3.9

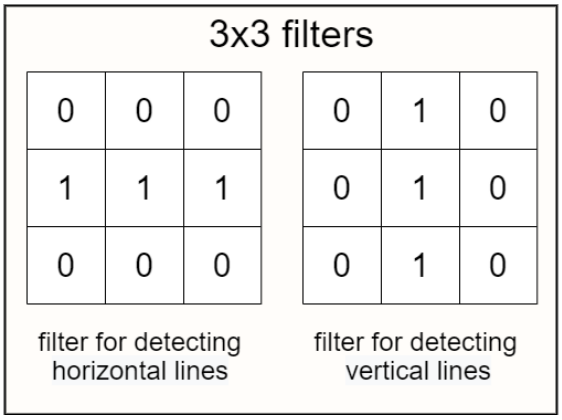

Figure 3.9: Different filter examples

In convolutional neural networks, numbers inside the filters represent the weights, and they are learned during the training with the help of optimization algorithms. This means that the neural network learns what kind of features it should extract from the image.

3.2.3 Fully Connected Neural Network and Convolutional Neural Network Comparison

Convolutional layer employs fewer different parameters than the Fully Connected layer since it forces input values to share the parameters, which is the primary distinction between the two layers. Every output is produced based on every input in the Fully Connected layer, which employs a linear procedure. The size of the small version of input which is subject to the filter's size determines the size of the output of the convolution layers, and the weights are distributed across all of the pixels. In other words, by using the nearby pixels as an input and the identical coefficients for all pixels, the output is created. The linear operation is done using the nearby pixels of the pixel where the filter center is located in the convolution. The adjacent pixels have a significant association already. We make the assumption that the nearby pixels serve as the primary proxies for the central pixel when employing the convolutional layer.

To summarize, Convolutions are not densely connected, which means that not all input nodes have an impact on all of the output nodes. Convolutional layers have more learning flexibility, which is very helpful when dealing with inputs that have high dimensionality like images. These benefits are what enable convolutional layers to learn features in the data, such as shapes like lines, curves, etc., in image data, which is a well-known attribute of the network. As a result of our prior understanding of our data and the information contained within it, we employ a convolutional layer to both lighten the burden on our model and to precisely point out the data that would be valuable for it to learn from while keeping it far from unnecessary data.

3.2.4 Pooling

A two-dimensional filter is slid over each channel of the feature map during the pooling operation, and the features contained within the area bounded by the filter are aggregated. After the convolutional layer, a new layer called a pooling layer is introduced. Particularly, following the implementation of a nonlinearity, such as ReLu, Tanh, Sigmoid, etc., to the feature maps produced by a convolutional layer. Each feature map is processed separately by the pooling layer, which produces a new set of the same amount of down-sampled feature maps. This implies that the size of each feature map will always be decreased by the pooling layer. Pooling is done by applying a filter. The size of the filter is less than the feature map. A shortened version of the features found in the input is produced by utilizing a pooling layer to obtain down-sampled

feature maps. The model's invariance to local translation refers to this ability that is added by pooling. Average Pooling and Max Pooling are two often used functions in the pooling process.

Average Pooling: For each region on the feature map, determine the average value.

Max Pooling: Determine the highest value for each feature map region.

3.2.5 Optimization Algorithms

We must alter each epoch's weights and minimize the loss function while training the deep neural network. An optimizer is a function or algorithm that alters the characteristics of a neural network, such as its weights/coefficients and learning rate. Consequently, it aids in the reduction of total loss and the improvement of accuracy. Because a deep neural network typically has millions of parameters, finding the proper weights for the model is a difficult issue. It is necessary that the selection of an appropriate optimization algorithm for the training of the neural networks.

Gradient Descent

A gradient gives us the steepest direction to reach the local minimum. First, start with initial weights, and calculates their loss. It starts searching for weights that can give lower loss values than the current loss. For this purpose, it moves for lower weights and updates these weights. The process continues until it obtains the local minimum or global minimum. There are 3 kind of gradient descent algorithms Figure 3.10.

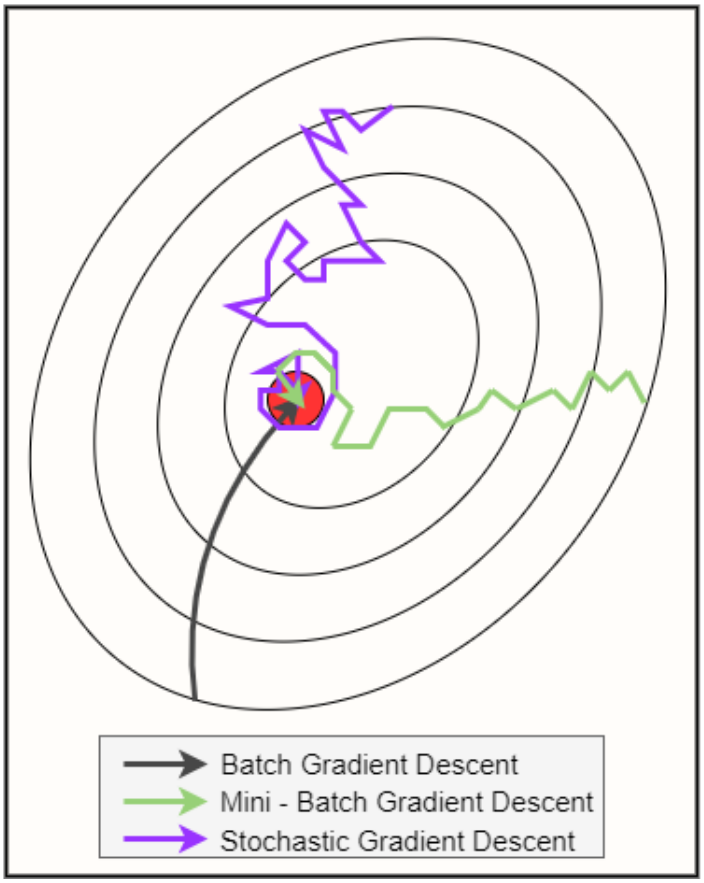

Figure 3.10: Batch - mini batch - stochastic gradient descent comparison

Batch Gradient Descent

It finds the gradient of the loss function depending on the weights (W) for the entire training dataset, where J is the loss function, and η is the learning rate which is the size of the steps to reach a local minimum or global minimum, $\frac{J}{W}$ is a gradient showing the direction in which the value W should be pushed in order to lower J Equation 3.5.

$$W = W - \eta \frac{\partial J(y, \hat{y})}{\partial W} \tag{3.5}$$

Stochastic gradient descent

Stochastic gradient descent (SGD) [31] update weights for each training sample $x^{(i)}$ with its label $y^{(i)}$ Equation 3.6.

$$W = W - \eta \frac{\partial J(y^{(i)}, \widehat{y}^{(i)}; x^{(i)})}{\partial W} \tag{3.6}$$

On big datasets, stochastic gradient descent converges quicker than gradient descent because updates are more frequent. Furthermore, because the data is frequently repetitive, the stochastic estimate of the gradient is usually precise without utilizing the entire dataset.

Mini-Batch Gradient Descent

For every mini-batch of n training samples, mini-batch gradient descent makes an update. This method reduces the variance of weight updates, resulting in more consistent convergence Equation 3.7.

$$W = W - \eta \frac{\partial J(y^{(i:i+n)}, \widehat{y}^{(i:i+n)}; x^{(i:i+n)})}{\partial W} \tag{3.7}$$

Momentum

Instead of relying solely on the gradient of the current step to drive the search, momentum considers the gradient of previous steps as well.

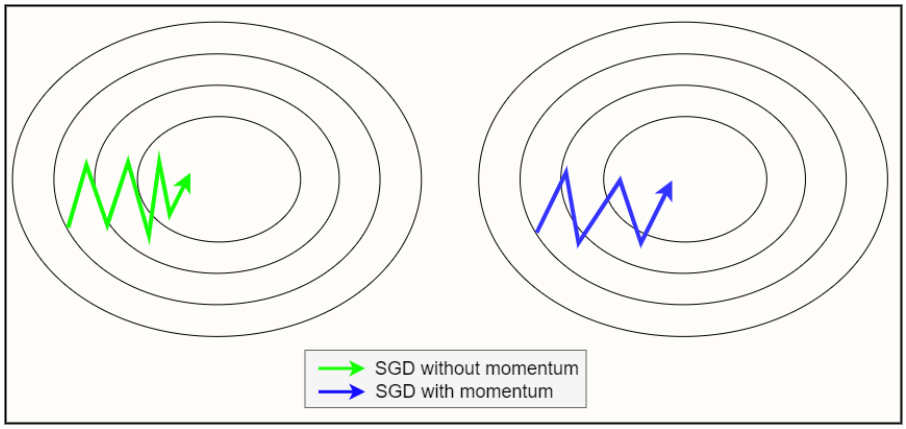

Figure 3.11: SGD with momentum and SGD without momentum comparison

As seen in Figure 3.11, momentum is a technique that aids in the speeding of SGD in the desired direction while also reducing fluctuations. It achieves this by adding a fraction of the previous time step's update vector to the current update vector Equation 3.8.

$$W = W - \eta * m_t \tag{3.8}$$

$$m_t = \beta * m_{t-1} + (1 - \beta) * (dW_t) \tag{3.9}$$

Where, $dW = \frac{\partial J}{\partial W}$ and β represents a parameter for the decaying average of parameters.

Adagrad

The Adaptive Gradient algorithm [32], often known as Adagrad, is a variant of the gradient descent optimization process. It adjusts the learning rate for each feature based on the problem's predicted geometry; in particular, it assigns greater learning rates to uncommon features, ensuring that parameter changes are based on significance rather than frequency. The Adagrad weight updating formula is Equation 3.10:

$$W_{t+1} = W_t - \eta * dW_t \tag{3.10}$$

At each iteration, α_t indicates different learning rates for each weight, where ε is a small pos-

itive number to prevent dividing by zero error as can be seen in Equation 3.11

$$\eta'_t = \frac{n}{\sqrt{\alpha_t + \varepsilon}} \tag{3.11}$$

α_t is represented as, where dW is derivative of loss with respect to weight which means that the amount of change in the loss value according to the weight value is as in the Equation 3.12

$$\alpha_t = \sum_{i=1}^{t} dW_{t,i}^2 \tag{3.12}$$

One of the main drawbacks of the Adagrad is α_t may grow high as the number of iterations grows, causing η'_t to decline at a larger rate. As a result, the old weight will be nearly equal to the new weight, resulting in slow convergence.

RMSprop

RMSprop, or root mean square prop, is an adaptive learning approach for improving AdaGrad. It uses the decaying average of parameters in each parameter's step size adaption instead of the cumulative sum of squared gradients, as AdaGrad provides Equation 3.13.

$$W_{t+1} = W_t - \eta' * dW \tag{3.13}$$

Where, $\eta' = \frac{\eta}{\sqrt{v_t + \varepsilon}}$ and v_t represented as Equation 3.14:

$$v_t = \beta * v_{t-1} + (1 - \beta) * (dW_t)^2 \tag{3.14}$$

Adam

Another method for calculating adaptive learning rates for each parameter is Adaptive Moment Estimation (Adam) [33]. Adam combines the best features of the Adagrad and RMSprop methods. Adam preserves an exponentially decaying average of past gradients comparable to momentum, in addition to the exponentially decaying average of the square of past gradients v_t like RMSprop. Adam computes the decaying averages of past gradients m_t and past squared gradients v_t respectively as here Equation 3.9 and Equation 3.14. The authors of Adam remark that m_t and v_t are biased to zero, specifically in the first time steps and when the decay rates are low because they are begun as vectors of zeros. The authors overcame these biases by comput-

ing first and second moment estimations. The authors overcome these biases issues by applying the following Equation 3.15 and Equation 3.16:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad (3.15) \qquad\qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad (3.16)$$

The following Equation 3.17 is the update rule:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\widehat{v}_t + \varepsilon}} * m_t \qquad (3.17)$$

3.2.6 Activation Functions

An artificial neural network can learn complex patterns in the data with the help of an activation function. The activation process ultimately chooses what signals should be sent to the following neuron. It receives the output signal from the neuron before it and transforms it into a form that could be used as the input for the neuron after it. The ability of an activation function to add non-linearity to the neural network is its most crucial characteristic. Additionally, they assist in limiting the value of the neuron's output to a predetermined level. This is significant since the activation function's input consists of the product of W and x then plus b, where W is weight, x is the input data, and b is the bias. In the case of deep neural networks, which include millions of parameters, this number, if not constrained to a specific limit, can reach very high magnitudes, and computational problems can cause because of this issue.

The output signal transforms into a simple linear function if the activation function is not used. Only polynomials of one degree are linear functions. Without an activation function, a neural network will behave like a linear regression with constrained learning capacity. However, we also want our neural network to learn nonlinear scenarios. Because we'll teach our neural network using complicated real-world data, including images, videos, texts, and audio. This allows multilayer deep neural networks to extract useful characteristics from the data. Nonlinear functions are those that have more than one degree. To accomplish this, artificial neural networks are intended to work with different types of functions. This calls for their ability to compute and learn any given function. Stronger learning of networks is possible due to the nonlinear activation functions. The backpropagation [25] methods are used in the artificial neural network to calculate the error values associated with the weights. The optimization technique must be chosen, such as stochastic gradient descent, and the loss must be reduced.

Linear: The weighted sum of the input is not altered by this function; it merely returns the

value that was passed to it as shown in Figure 3.12. It undoubtedly enables the connection of many neurons. But the value of its derivative is constant. Because the function's derivative is a constant and has no connection to the input x, using backpropagation is not possible.

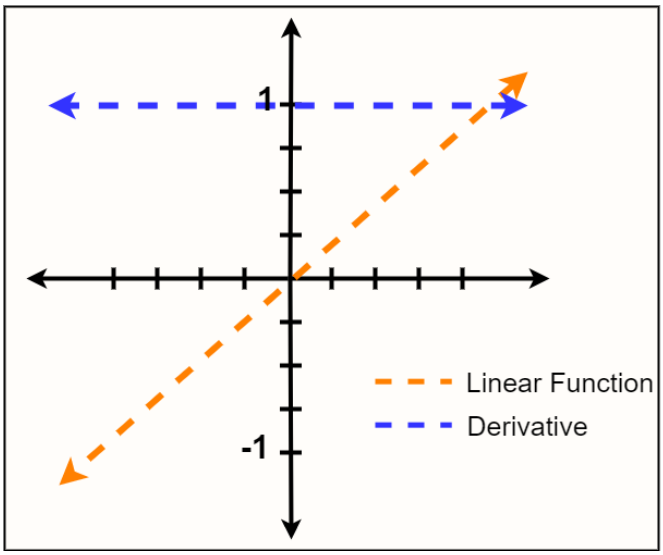

Figure 3.12: Linear activation function visualization

ReLu: The ReLU or Rectified Linear Unit function [34] is a very common activation function used in the hidden layers of artificial neural networks. ReLU takes values in the range of $[0, +)$ Equation 3.18. In an artificial neural network with many neurons and hidden layers, sigmoid and tanh cause almost all neurons to fire or activate in the same way. This means that all signals will be sent to the following neurons. As a result, these activations are labor-intensive and necessitate extensive processing. Some activations can be diluted for a more efficient computational intensity. It is possible to achieve this state with the ReLU activation function. Getting 0 values on the negative axis also means that the network will run faster. Since the amount of computation is less than the sigmoid and hyperbolic tangent functions, it is preferred in multilayer networks. However, since the derivative of the 0 regions is 0, learning does not take place in this region can be seen in Figure 3.13. This is one of the disadvantages of ReLu.

$$f(x) = max(0, x) \tag{3.18}$$

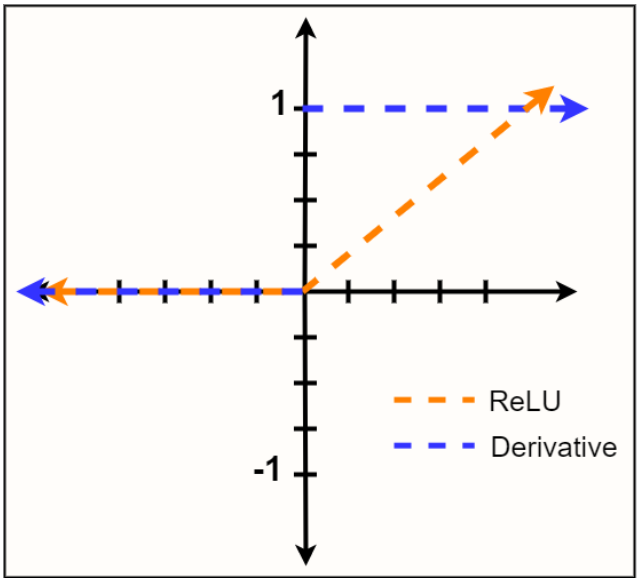

Figure 3.13: ReLu activation function visualization

Softmax: It has a structure very similar to the sigmoid function. It performs quite well when used as a classifier, just like Sigmoid. Softmax [35] can be used in binary classification as in Sigmoid, as well as in multiclass classification problems. It determines the probability of the input belonging to a certain class by generating values in the range of $0 - 1$ as represented by the Equation 3.19.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3.19}$$

3.2.7 Cross Validation

Cross-validation is a statistical method to improve the ability of machine learning models. The cross-validation technique is used especially when there is a limited amount of data to improve results. In applied machine learning, the cross-validation technique is commonly used to reduce bias and prevent overfitting. When a machine learning model learns the noise of the data, overfitting occurs, as shown in Figure 3.14. Overfitting, as the term suggests, arises when a model or algorithm fits the data too much. A model that has been overfitted produces good accuracy on training data but subpar outcomes on new data.

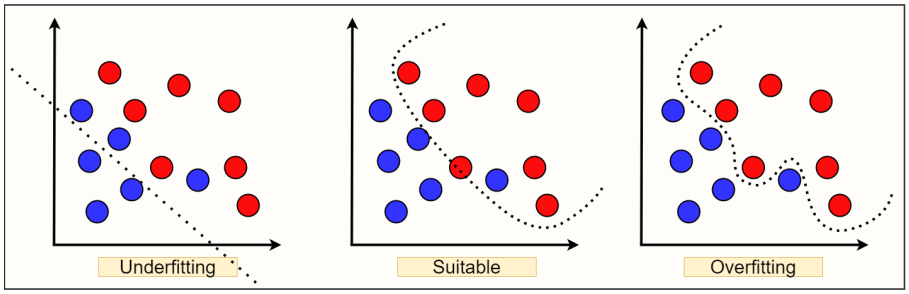

Figure 3.14: Overfitting and underfitting issues visualization

Cross-validation is generally applied as follows:

- Shuffle the dataset,
- Divide the dataset into k subgroups,
- For each subgroup;
  - Treat one of the subgroups as the test dataset to validate the model,
  - Treat the rest of the subgroups as the training dataset,
  - Fit the model on the training dataset, then evaluate it on the test dataset,
  - Keep learned knowledge and use it in the next subgroup,

- Summarize the performance of the model using each evaluation score.

3.3 Transfer Learning

In machine learning problems, transfer learning refers to the reuse of a previously trained model on a new problem Figure 3.15. Transfer learning applies what it is learned from a previous assignment to improve the accuracy of new task predictions. Transfer learning can yield much higher performance than training with a small amount of data when applied to a new task. It can save time and resources by utilizing pre-trained models rather than having to train models from scratch to perform similar tasks. It can also help you overcome a lack of labeled training data. The following is a list of items that can be used to summarize the reasons for using transfer learning:

- To avoid having to train several machine learning models from scratch to fulfill similar tasks, to save resources and time.
- A broader approach to problem solving that uses a variety of algorithms to address new problems.
- Using pre-trained models, an organization or an individual can overcome the limitation of labeled training data.
- As a cost-effective way to save in areas of machine learning where large quantities of resources are required, such as image classification or natural language processing.
- Each new model no longer requires a big collection of labeled training data.
- Increasing machine learning development and implementation efficiency.
- Models can be trained in simulations rather than in real-world settings.

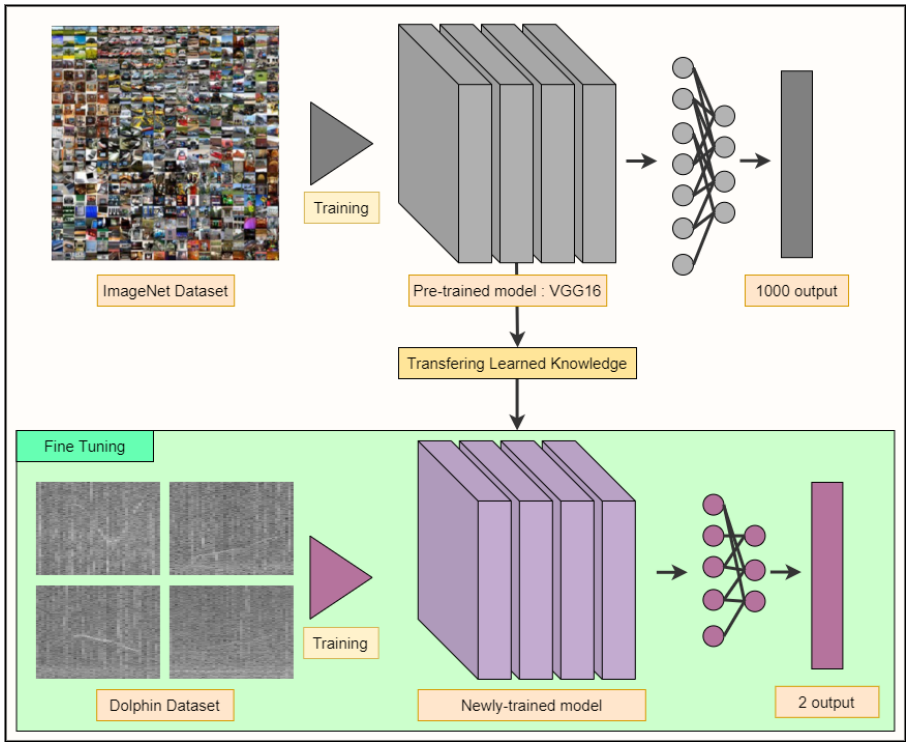

Figure 3.15: Transfer learning representation

3.3.1 VGG16

The VGG16 [16] convolutional neural network model was utilized to win the 2014 ILSVR ImageNet [36] competition. ImageNet contains about 1.2 million images for training with 1000 classes. It is the smallest depth model built into Keras applications. Keras Applications are deep neural network models that come with weights that have already been trained. Prediction, feature extraction, and fine-tuning are all possible by using these models. On the ImageNet validation dataset, all of the models in Keras applications have a top-5 accuracy of more than 89%. VGG16 has a 90.1% accuracy rating in the top five. Instead of a huge number of hyper-parameters, VGG16 concentrated on having 3x3 filter convolution layers with stride 1 and used the same padding and max-pooling layer of 2x2 filter with stride 2. Throughout the architecture, it maintains this convolution and max-pooling layer structure. Finally, it has two fully connected layers (dense layer) for output, followed by softmax. The 16 in VGG16 alludes to the fact that it comprises 16 layers, each of which has weights, as can be seen in Figure 3.16.

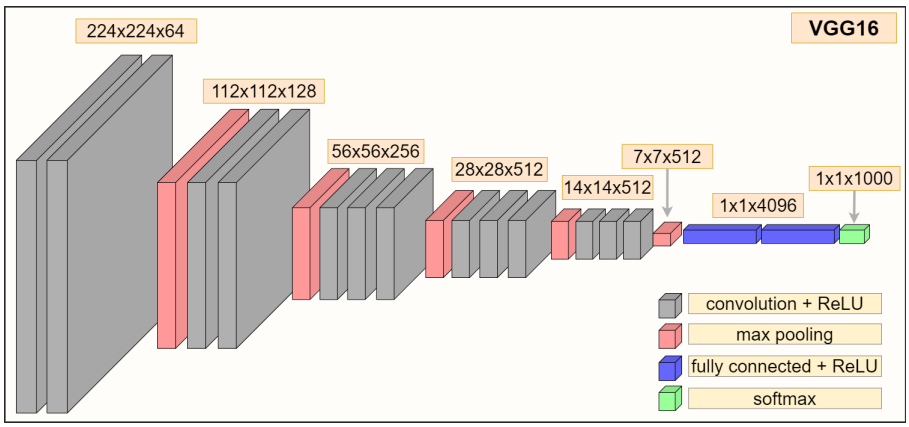

Figure 3.16: VGG16 architecture

3.4 Regularization Methods

3.4.1 Data Augmentation

Data augmentation refers to a set of approaches for generating new training samples from existing ones by applying random oscillations and fluctuations while maintaining the data's labels. Making modest modifications to data or utilizing deep learning models to produce additional data points are examples of this. When using data augmentation, the goal is to improve the

model's generalization. The network may learn more robust characteristics since it is constantly exposed to new, slightly tweaked copies of the input data Figure 3.17. Data augmentation is not used during testing and instead evaluates the trained network on the raw testing data. In most circumstances, it is noticed that a gain in testing accuracy is at the expense of a tiny drop in training accuracy. Machine learning models can benefit from data augmentation to improve their performance and outcomes. A machine learning model works better and more correctly when the dataset is rich and sufficient.

Data collection and labeling can be a very time-consuming problem and expensive for machine learning models. Organizations or individuals can lower these costs via transforming datasets utilizing data augmentation approaches. Cleaning data is one of the phases of creating a data model, and it is required for highly accurate models. Therefore, if data cleaning limits the representativeness of the sample training data to the real-world data, the model will be unable to make accurate predictions for real-world data. We can make machine learning models more robust via data augmentation approaches, which form variances that the model can be encountered in the actual world. The following are examples of traditional image processing activities for data augmentation:

- Rotating,
- Rescaling,
- Vertical and horizontal flip,
- Padding,
- Translation,
- Cropping,
- Zooming,
- Adding noise,
- Erasing,
- Color modifications such as darkening and brightening,
- Grayscaling.

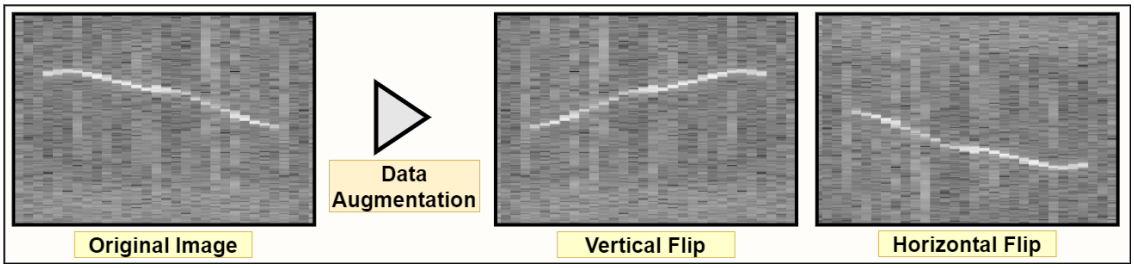

Figure 3.17: Data augmentation example via dolphin whistle spectrogram

The following are some of the advantages of data augmentation:

- Improving the accuracy of the model prediction.
- Increasing the capacity of the model to generalize.
- Creating data heterogeneity and reducing overfitting of the model.
- Lowering the cost of data collection and labeling.
- Assisting in settlement of concerns of class imbalance in classification.
- Allowing for the prediction of unusual conditions.
- Preventing issues with data privacy.

3.4.2 Dropout

A regularization technique called dropout which generalizes the training of a deep neural network with various architectures. A certain proportion of layer outputs are dropped out or ignored in a random way during the training as seen in Figure 3.18. This means that the layer appears to have different nodes and connections to the previous layers. Dropout has the effect of making training noisy, pushing nodes within a layer to assume more or less responsible for the inputs based on probabilities which are genuinely beneficial to prevent overfitting problems. As a result of the dropout method's ability to simulate numerous networks, these networks have nodes that are generally more robust to inputs.

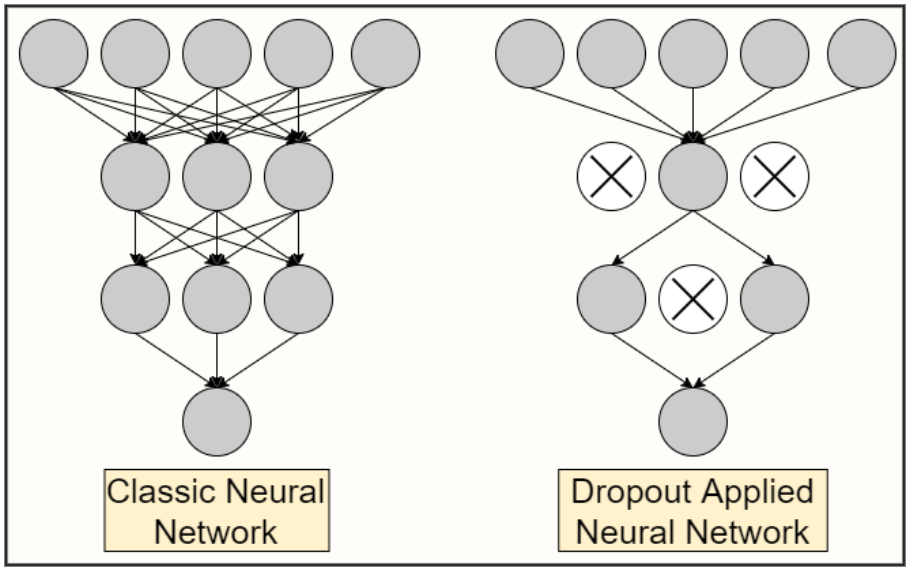

Figure 3.18: Dropout visualization

4 Tools

4.1 Keras

Keras is a Python-based application programming interface (API) for deep neural networks. This open-source neural network framework is built on top of TensorFlow and can be used to experiment with deep neural networks quickly. Keras emphasizes modularity, usability, and extensibility. In the middle of 2017, Keras was adopted and integrated into TensorFlow. The tf.keras module gives users access to it. The Keras library, on the other hand, can continue to function independently. Over Keras executing low-level tensor operations efficiently on the CPU, GPU, or TPU is possible, and the gradient of differentiable functions can be computed. It allows dividing computations to a large number of devices, such as several GPUs in a cluster. Also, It is possible to export programs to servers, browsers, mobile devices, and embedded devices.

4.2 TensorFlow

TensorFlow is a Google-developed open-source deep learning framework that was unveiled in 2015. It is well-known for its documentation and training assistance, as well as its flexible production and deployment choices and support for a variety of platforms, including web apps, mobile apps, and IoT devices. TensorFlow is a framework that is a rapidly-growing introduc-

tion into the field of machine learning, with complete community resources, frameworks, and tools that make developing and deploying machine learning products easier. TensorFlow has also integrated Keras.

4.3 TensorFlow - Keras Comparison

Keras is a neural network library that works on top of TensorFlow. TensorFlow is an open-source end-to-end platform and library for different machine learning tasks. Both offer high-level APIs for quickly creating and training models, while Keras is much more user friendly due to its Python integration.

Because Keras is a library inside TensorFlow's framework, a deep neural network model can be created using the Keras interface, which is easier to implement, and then dropped into TensorFlow when a feature that Keras lacks or a particular TensorFlow feature is required. As a result, TensorFlow code can be directly integrated into the Keras training process.

When dealing with massive datasets and object identification, researchers resort to TensorFlow for its high functionality and performance. Linux, macOS, Windows, and Android are all supported by TensorFlow. Google Brain created the framework, which is now used for Google's research and development purposes.

4.4 Keras Image Data Generator

Keras ImageDataGenerator takes the raw data as input, transforms it at random, and returns an output consequent containing solely the recently changed data. It does not add more data to the original dataset. The Keras image data generator is also used for data augmentation, with the goal of improving the model's generalization, and it allows to use in the realm of real-time data augmentation to generate batches. When we utilize the image data generator, we can iterate through the data in batches. In data augmentation, operations, including rotations, translations, scale adjustments, and horizontal-vertical flips, are performed at random using an image data generator.

4.5 Optuna

Optuna is a software framework for hyperparameter optimization that was created specifically for machine learning pipelines. Users of Optuna can employ methods for sampling hyperpa-

rameters and removing unsuccessful trials. Comparing this to more conventional techniques like GridSearch, it substantially speeds up optimization time and performance. Additionally, it enables users to visualize optimization histories to comprehend the model better. Optuna chooses which hyperparameter values to evaluate next based on a historical record of trials. Using the historical information of the previously finished trials, Optuna repeats this process. More specifically, it uses the Tree-structured Parzen Estimator, a Bayesian optimization algorithm.

5
Problem Formulation

We aim to develop an efficient model that performs automatic dolphin whistle detection. For this purpose, spectrograms containing dolphin whistles and spectrograms that do not contain not dolphin whistles were obtained by using the 243 underwater records we have, and a training/testing dataset was created. Our problem is to detect the spectrograms containing the dolphin whistle by scanning the audio file with a fixed window of the specified length, which shows that our problem is a binary classification problem. Spectrograms without dolphin whistles represent negative class and are labeled 0, while those with dolphin whistles represent positive class and are labeled 1. In this direction, our deep learning and machine learning models were trained and tested with the data set we created.

5.1 Supervised Learning

The algorithm's learning phase is completed given a dataset containing some observations and their labels/classes. For instance, images of animals might be observed, with labels indicating the animal's names, such as cat, dog, bird, etc. Regression and classification cases are two types of supervised models.

Regression: Regression is a statistical process to estimate the relationships between a dependent variable Y, which is called output, and one or more independent variables X, which are called features. In regression problems, the output variable is a continuous value such as finan-

cial forecasting, stock price prediction, sales, and promotions forecasting, etc.

Classification: When the output of the algorithm is a category, the problem is called a classification problem. As in our problem, ”there is a dolphin whistle” and ”there is no dolphin whistle” are examples of these categorical outputs.

5.1.1 Binary Classification

Binary classification is a problem that classifies the components of a set into two bunches called classes/labels. For binary classification issues, medical testing to decide if a malady exists or not, deciding whether an email is spam mail or not spam mail, and finding whether a user visiting an online store buys or does not buy a product can be given as examples. Our problem is also a binary classification problem. Our aim is to build up a highly accurate model that automatically detects whether spectrograms created from sounds recorded from underwater contain dolphin whistles or not contain. For this purpose, we used and experienced different methods, such as support vector machines, logistic regressions, and convolutional neural networks.

5.2 Dataset and Preprocessing

The dataset was created by generating spectrograms using 243 records, each 5 minutes long, provided by the University of Haifa. Each window is 0.8 seconds long. Spectrograms were obtained by shifting the window on the record for 0.4 seconds, as seen in Figure 5.1. With this method, it is aimed that 1 window can fully cover 1 signal. While recording the spectrograms, the low and high frequencies were cut, and the spectrogram creation was completed in this way. Low frequency limit is determined as 3 kHz, high frequency limit is determined as 20 kHz.

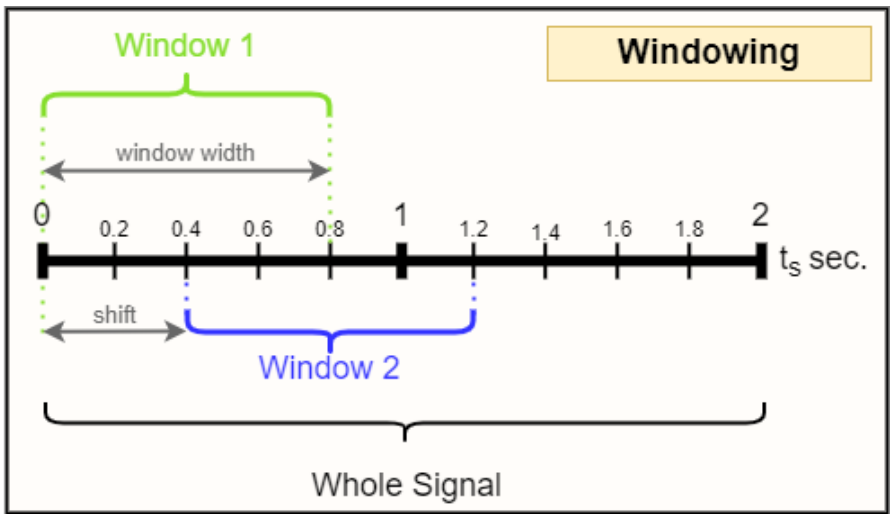

Figure 5.1: Windowing process of recordings taken from the underwater

5.2.1 Signals and Features

The audio recordings were recorded during the month of June 2021. All sound recordings are 5 minutes long and have 2 channels. Each audio file is formatted in the flac file type. Flac file format allows you to store data without data loss. 1 second of each record, which is 5 minutes long, contains 96000 samples.

5.2.2 Data Preprocessing

First of all, outliers in the signal lengths were calculated using the information of dolphin voices that were previously tagged in seconds and frequency ranges to determine the window size in order to create the spectrograms. The calculation was made using 795 tagged data. In the analysis, signals longer than approximately 0.78 seconds were determined as high outliers, and signals shorter than approximately 0.14 seconds were determined as low outliers. This calculation was made with the help of the Tukey method [37].

The method finds the outliers based on the quartiles of the data. Firstly, the first quartile, second quartile, which is the median value, and the third quartile of the data are calculated. The first quartile, or Q1, represents 1/4 of the data, the second quartile, or Q2, represents 1/2 of the data, and the third quartile, or Q3, represents 3/4 of the data. Next, the calculation of the interquartile range IQR follows. According to the method, the outliers are values that are more

than 1.5 times the interquartile range IQR from the quartiles, as shown in the Equation 5.1.

$$\begin{aligned} IQR &= Q_3 - Q_1 \\ High\ Outlier &= Q_3 + 1.5IQR \\ Low\ Outlier &= Q_1 - 1.5IQR \end{aligned} \tag{5.1}$$

As a result of the calculations, the window size was determined as 0.8 seconds to cover dolphin sounds. Audio files consist of 2 channels, and before spectrograms are created, the average of 2 channels is taken, and in this way, it is aimed at reducing noise Figure 5.2.

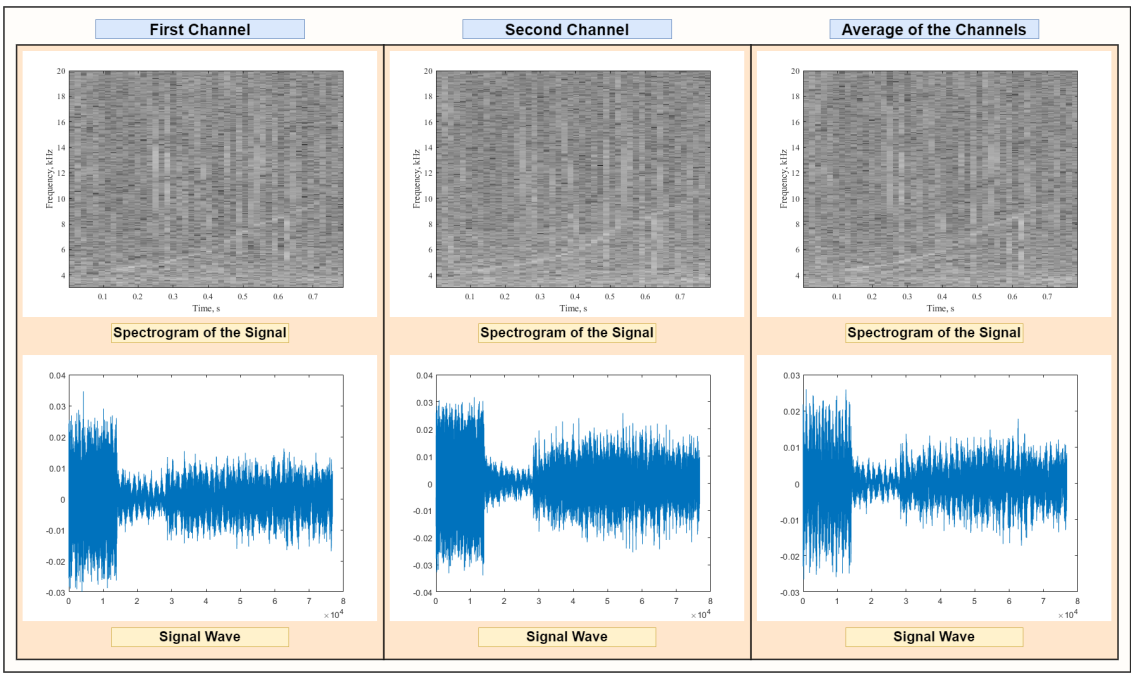

Figure 5.2: Visualization of different channels of the recordings taken underwater

Spectrograms of the dolphin whistles are created via using MATLAB's spectrogram function from the digital signal processing toolbox. The spectrogram function calculates the short-time Fourier transform of a signal, and it returns complex numbers, which include both magnitude and phase information. Since each created spectrogram represents a 0.8 seconds window, 76800 elements from the signal are used in each spectrogram. As the window, the Blackman function with 2048 points and periodic sampling method is used. The number of overlaps is determined as 20% of the Blackman window length which is 410. FFT length is used as 2048,

while the sampling frequency is determined to be 96000 Hz.

The output of the spectrogram function is used to visualize the spectrograms in the imagesc function of the digital signal processing toolbox. Before visualizing, the frequency output of the spectrogram function is converted to kHz from Hz. Also, the power spectrum density is converted into dB to be used in the imagesc function. Later time instants returning from spectrogram function are used with the frequency output and converted power spectrum density to create images with imagesc function. y-axis limits of the imagesc function are set between 3 and 20 kHz since the dolphin whistles are often existing between this interval. Gray colormap is used while visualizing.

5.2.3 Dataset Details

The training dataset contains 4198 spectrograms. Half of them are positive samples containing dolphin sounds, and the other half are negative samples that do not contain dolphin sounds. To test the training results, a non-balanced test data set containing 300 positive samples and 4700 negative samples was used. Here, it is desired to observe the results of the models on the non-balanced data. The spectrograms used for training and testing were collected from the 243 records each of them 5 minutes long reserved for it.

5.3 Evaluation Metrics

Metrics for evaluation are used to assess how well a statistical or machine learning model is performing. Every machine learning workflow has evaluation metrics as a component. The ability of evaluation metrics to distinguish between different model outcomes is a crucial feature. It is crucial to evaluate the corresponding model using a variety of evaluation metrics. This is due to the fact that a model may perform well when using a measurement from one evaluation metric but poorly when using a different measurement from a different evaluation metric. In order to make sure the model is working properly and ideally, evaluation metrics are essential.

5.3.1 Loss Functions

The loss function determines how far the algorithm's current output is from what is desired. This is a technique for assessing how well the algorithm fits the data. There are 2 different categories, one of them is for classification, which is for discrete values, and the other one is for

regression, which is for continuous values. In short, a loss function evaluates how inaccurate the model is in estimating the relationship between input and output.

Cross-entropy: The purpose of this function, which derives from information theory, is to compare two averages of the distribution's amount of data. The difference between two probability distribution functions is calculated using the cross-entropy. Binary and multiclass classification issues can be solved using cross-entropy.

Binary cross-entropy: Binary cross-entropy can be used for binary classification. Equation 5.2 is the equation of binary cross-entropy.

$$-(ylog(p) + (1-y)log(1-p)) \tag{5.2}$$

Categorical cross-entropy: This function can be used for both binary and multiclass classification. Labels have to be encoded as categorical such as one-hot encoding, as shown in Figure 5.3. Equation of categorical cross-entropy can be seen in Equation 5.3

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{5.3}$$

Where M represents the number of classes, y is the binary indicator and, p is the predicted probability observation o of class c.

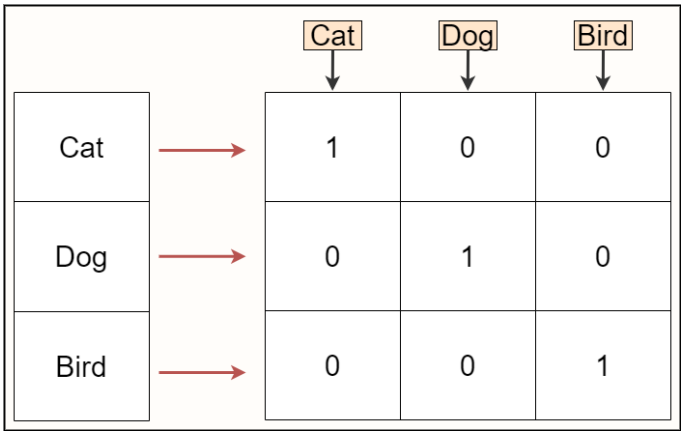

Figure 5.3: Example one hot encoding implementation

Sparse cross-entropy: This loss function can be used for binary and multiclass classification problems. The labels have to be an integer number between o and n depending on the number of classes.

Log-Loss: The binary version of cross-entropy utilized in classification tasks is this function. It evaluates the efficiency of a classification model, the result of which is a probability with a range of 0 to 1.

Mean Absolute Error (MAE): L1 regularization referred to as mean absolute error, is employed in regression tasks. It computes the ratio of the number of outputs divided by the absolute difference between the current output and the expected output. Unlike Mean Square Error, which is vulnerable to outliers, Mean Absolute Error is focused on absolute values. MAE equation can be seen as Equation 5.4:

$$\frac{1}{m}\sum_{i=1}^{m}\left|y^{(i)}-\hat{y}^{i}\right| \tag{5.4}$$

Where m represents the number of samples, y is the true label and $\hat{y}$ is the predicted label.

Mean Square Error (MSE): This procedure, also referred to as L2 regularization, is employed in regression problems. It describes the degree to which a regression line resembles a set of data points. It determines the output number divided by the square of the difference between the present output and the expected output, as can be seen in Equation 5.5. However, because it uses the square difference, Mean Square Error loss is more susceptible to outliers.

$$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{i})^{2} \tag{5.5}$$

5.3.2 Confusion Matrices

A table known as a confusion matrix is frequently used to explain how well a classification model performs on a set of test data when the true labels are known. It is a highly well-liked tool for tackling classification problems. Both binary classification and multiclass classification tasks can benefit from it. The confusion matrix for a binary classification task is shown in Figure 5.4.

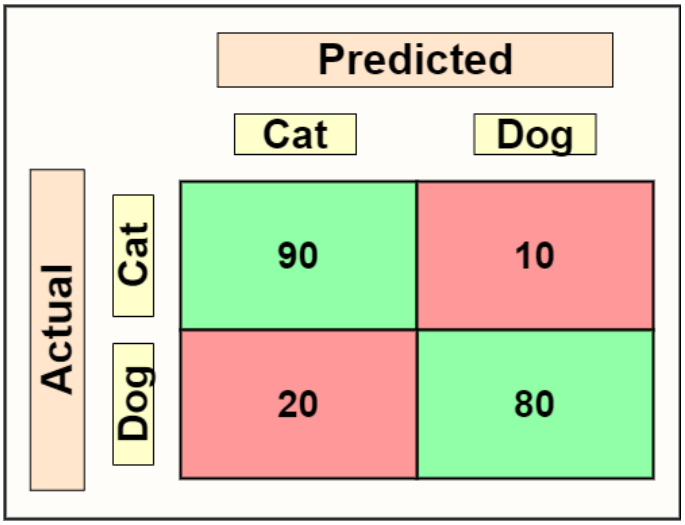

Figure 5.4: Confusion matrix example

From Figure 5.4, we can understand several things;

- There are two possible predicted classes: "cat" and "dog".
- The classifier made a total of 200 amount of predictions (e.g., 200 animals were being tested for their animal type).
- Out of those 200 cases, the classifier predicted "dog" 90 times and "cat" 110 times.
- In reality, 100 animals in the sample are dogs, and 100 animals are cats.

In a two-class situation, such as the one we investigate in this work, we frequently seek to distinguish between observations with a certain outcome and regular observations. For example, a disease state or event that was not a disease state or event. We can designate the event row as "positive" and the row with no event as "negative". This gives us:

- "true positive (TP)" for correctly predicted event values.
- "false positive (FP)" for incorrectly predicted event values.
- "true negative (TN)" for correctly predicted no-event values.
- "false negative (FN)" for incorrectly predicted no-event values.

The accuracy for the confusion matrix in Figure 5.4 is 85% which is calculated with the Equation 5.6.

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total\ Sample} \tag{5.6}$$

5.3.3 Receiver Operator Characteristic Curves

A measurement tool for binary classification issues is the Receiver Operator Characteristic (ROC) curve. It is a probability curve that, in essence, separates the "signal" from the "noise" by plotting the True Positive Rate (TPR) versus the False Positive Rate (FPR) at different threshold values.

True Positive Rate (TPR): It is also known as "Sensitivity" or "Recall". TPR is the probability that an actual positive will be predicted positive. It is calculated as Equation 5.7.

$$TPR = \frac{TP}{TP + FN} \tag{5.7}$$

False Positive Rate (FPR): It's the probability that a false alarm will be raised which means a positive outcome will be given when the true value is negative. It is the probability that a false alarm will happen, meaning that a positive result will be reported when a negative value actually exists. It is calculated as Equation 5.8

$$FPR = \frac{FP}{TN + FP} \tag{5.8}$$

The capacity of a classifier to differentiate between classes is measured by the Area Under the Curve (AUC), which is used as a summary of the ROC curve. The model performs better at differentiating between the positive and negative classes the bigger the Area Under the Curve. AUC = 1 indicates that the classifier can accurately distinguish between all Positive and Negative class points. The classifier would be predicting all Negatives as Positives and all Positives as Negatives, if the AUC had been 0. The classifier cannot discriminate between Positive and Negative class points when AUC=0.5. This indicates that the classifier is either forecasting a random class or a constant class for each data point. There is a good possibility that the classifier will be able to tell the difference between the Positive class values and the Negative class values when the AUC is 0.5<AUC<1. This is the case because more True positives and True negatives can be detected by the classifier than False positives and False negatives. The ROC curve is presented in Figure 5.5 with TPR versus FPR, with TPR on the y-axis and FPR on the x-axis.

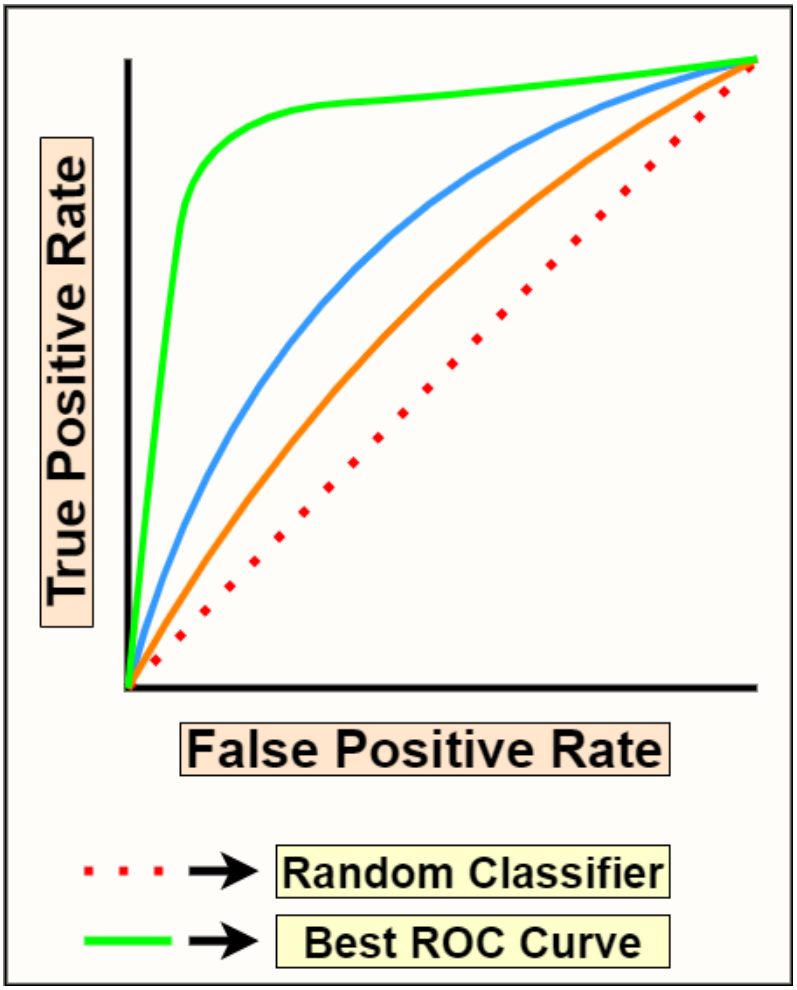

Figure 5.5: Roc curve example

6

Detection methods

6.1 Machine learning Models

After we created the spectrograms and created our dataset, spectrograms that we called negative without dolphin whistles were labeled 0, and spectrograms that included dolphin whistles were labeled as positive 1. The training results were completed using 4 different architectures. These architectures are pre-trained model VGG16 [16], vanilla CNN, SVM, and Logistic regression. In the training of our models, k-fold cross-validation was applied, and the number of folds was determined as 8. The K fold cross-validation method is made using the StratifiedK-Fold method of the scikit-learn library for the Python programming language. This method ensures that each validation set contains an equal number of classes of data. That is, each validation set is balanced in terms of the label. Models were also trained using an equal amount of validation sets without applying the cross-validation technique. Only real-time data augmentation has been applied to the training dataset. In this way, it is aimed to improve the results. The results were obtained for SVM and Logistic regression without cross-validation and with cross-validation. For VGG16 and vanilla CNN, as seen in Figure 6.1, cross-validated and non-applied versions, as well as data augmented and non-data-augmented versions, were trained for comparison. To have a fair comparison, the data are always separated at the same randomness using the same seed parameter, and the training without cross-validation was also performed using the first fold validation set out of 8 folds as the validation set. Data augmentation was

performed using ImageDataGenerator from the TensorFlow Keras library, and only horizontal flip and vertical flip methods were applied. The early stop method is used in both deep learning models, and patience is set to 15. The maximum number of epochs is 100, and the batch size is 32 in both models.

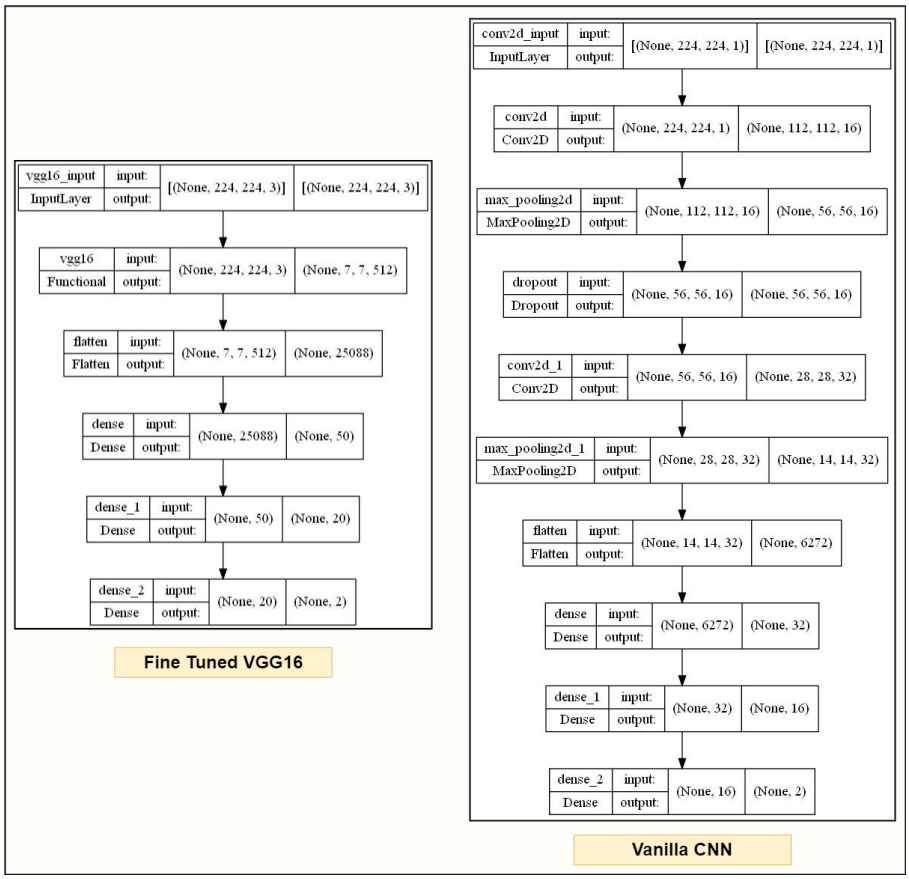

Figure 6.1: Neural networks architectures

The spectrograms have been resized to 224x244 because the size of the input tensor accepted by VGG16 is 224x224x3. 3 represents the RGB channel. The spectrograms we created are grayscale images, which are 2D arrays. However, since the size of the input tensor accepted by VGG16 is 224x224x3, the 2D image array was converted to 3D. This process was carried out by duplicating the same image array. Before the training, the preprocess_input function of VGG16, which was built in Tensorflow, was applied to the data set. This function converts images from RGB to BGR and makes the data zero-centered with respect to the ImageNet [36] dataset. After our data is zero-centered with the help of this function, the training is done.

The top part of VGG16 was deleted, 2 fully connected layers were added, and the number of nodes included in the layers was 50 and 20, respectively. ReLu activation function [34] is used in these layers. The output layer is Softmax. The Adam [33] optimization algorithm was used with a learning rate of 0.00001. Binary cross-entropy is implemented as a loss function, and the trainable parameter is set to True.

We wanted to create a CNN model with several layers and compare it with the pre-trained model VGG16. For our CNN and other machine learning models, the training was carried out by resizing the images as 224x224 with the aim of making a logical and fair comparison. In addition, before the training, all data pixels were normalized by dividing by 255. Our CNN model includes 2 convolution layers. Between these two layers, there are max pooling and dropout layers, respectively. After the last convolution layer, the max-pooling layer is applied, followed by 2 fully connected layers with the ReLU activation function. These two fully connected layers contain 32 and 16 nodes, respectively. As in VGG16, Softmax is used in the last layer. In the Convolution layers, 16 and 32 kernels are used, respectively, and the kernel size is (7,7) and (5,5). The stride value was determined as 2 in max-pooling layers and convolution layers, and the linear activation function was used as the activation function. The dropout rate of the dropout layer is set to 0.2. The Adam optimization algorithm was used with a learning rate of 0.0001. As a loss function, binary cross-entropy has been applied as in VGG16. These hyperparameters are tuned using Optuna, a hyperparameter optimization framework.

L2 Regularization technique was used for the penalty parameter in Logistic Regression, and the value of the C parameter was determined as 10. For SVM, while the C parameter is used as 10 as in Logistic Regression, RBF is chosen as the kernel function. These selections were made using the GridSearchCV method in scikit-learn.

6.2 Processing pipeline

Firstly, the dataset was created by collecting positive and negative samples from 243 records. Positives represent spectrograms that contain dolphin sounds, and negatives represent spectrograms that do not contain dolphin sounds. The length of the spectrogram windows in seconds is determined as 0.8 seconds. This window length was determined after the lengths of the dolphin signals were analyzed. The analysis was made with the Tukey method. When we ignore the outlier data found as a result of the analysis, the longest signal was calculated as approximately 0.78 seconds. For this reason, the window length is set as 0.8 seconds to fit 1 signal into a window. Spectrograms were recorded with 0.8-second windows, shifted for 0.4 seconds on

the record, and positive/negative samples were collected. Determining the window-shift time is to reduce the number of windows containing the same signals while at the same time providing the environment where 1 window can fully cover 1 signal. Secondly, The spectrograms created were used to feed VGG16, a pre-trained network on ImageNet, vanilla CNN, Logistic regression, and SVM algorithms. While the results obtained with VGG16 gave remarkably good results, efficient results were not obtained with vanilla CNN, SVM, and Logistic Regression. Here Figure 6.2 is the demonstration of the dolphin whistle detection pipeline we created.

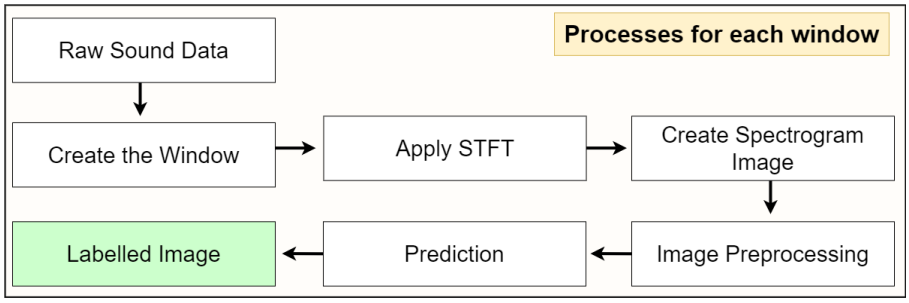

Figure 6.2: Processing pipeline representation

7

Experimental Results

Spectrograms of the dolphin whistles are created using MATLAB. Our work was developed in the environment of NVIDIA GeForce GTX 950M graphics card, Intel Core i7-6700 HQ processor, and 8 GB of RAM using TensorFlow 2.8. Saving of spectrograms was done using MATLAB software, and about 2-3 spectrograms per second can be saved with the above hardware. The labeling section by separating the spectrograms as positive and negative is done using Python, and the image separation amount per second is about 15-20 images. The Python script also outputs a CSV file that returns the initial and finish points of the windows that contain dolphin whistles and their confidence, as can be seen in Figure 7.1.

	A	B	C	D
1	file_name	initial_point	finish_point	confidence
2	2021_06_24_22_10_34_171.flac	10	10.8	0.9392962
3	2021_06_24_22_10_34_171.flac	122	122.8	0.9994481
4	2021_06_24_22_10_34_171.flac	14.8	15.6	1
5	2021_06_24_22_10_34_171.flac	28.4	29.2	1
6	2021_06_24_22_10_34_171.flac	28	28.8	1
7	2021_06_24_22_10_34_171.flac	73.6	74.4	1
8	2021_06_24_22_10_34_171.flac	74	74.8	1
9	2021_06_24_22_30_35_175.flac	14.4	15.2	0.9926105
10	2021_06_24_22_30_35_175.flac	199.6	200.4	0.99967825
11	2021_06_24_22_30_35_175.flac	217.2	218	0.9999999
12	2021_06_24_22_30_35_175.flac	217.6	218.4	0.9404071
13	2021_06_24_22_30_35_175.flac	223.6	224.4	1
14	2021_06_24_22_30_35_175.flac	265.6	266.4	0.82640237
15	2021_06_24_23_35_45_188.flac	160	160.8	0.9917773
16	2021_06_24_23_35_45_188.flac	25.2	26	0.987362
17	2021_06_24_23_35_45_188.flac	98.8	99.6	0.9873841
18	2021_06_24_23_40_45_189.flac	1.2	2	0.68416995
19	2021_06_24_23_40_45_189.flac	140	140.8	0.99945265
20	2021_06_24_23_40_45_189.flac	61.6	62.4	0.7477432
21	2021_06_24_23_40_45_189.flac	80.8	81.6	0.9999981
22	2021_06_24_23_40_45_189.flac	86.4	87.2	0.98862106

Figure 7.1: Examples of our model's output

From our training results, we found that VGG16 [16] gave better results with a clear difference in the test dataset, which contains 300 positive and 4700 negative samples. We observed that the cross-validation technique would generally affect the results better. While applying data augmentation improved the results with VGG16, it was not very effective in vanilla CNN training. As we can see in Table 7.1 and Figure 7.2, the VGG16 trained by cross-validation and data augmentation gives the best results.

Table 7.1: Results of our deep learning models

Method	Cross Validation	Augmentation	Val Loss (avg for folds)	Val Accuracy (avg for folds)	Test Loss (avg for folds)	Test Accuracy (avg for folds)	Model Size (KB)
Fine-Tuned VGG16	Yes	Yes	0,0091	0,9971	0,0758	0,989	187.280
Fine-Tuned VGG16	Yes	No	0,011	0,9969	0,084	0,9882	187.280
Fine-Tuned VGG16	No	Yes	0,0302	0,9866	0,0667	0,9824	187.280
Fine-Tuned VGG16	No	No	0,1033	0,9714	0,1694	0,9496	187.280
Vanilla CNN	Yes	Yes	0,216	0,9176	0,9545	0,6872	2572
Vanilla CNN	Yes	No	0,1634	0,9445	0,8898	0,6992	2572
Vanilla CNN	No	Yes	0,4998	0,7561	0,7325	0,6546	2572
Vanilla CNN	No	No	0,4908	0,7599	0,6948	0,6694	2572

In CNN, on the other hand, while the cross-validation and non-data augmented version gave the highest accuracy, a decrease in the detection rate of positive samples according to the cross-validation and data augmentation applied versions was noticed as can be seen in Figure 7.2.

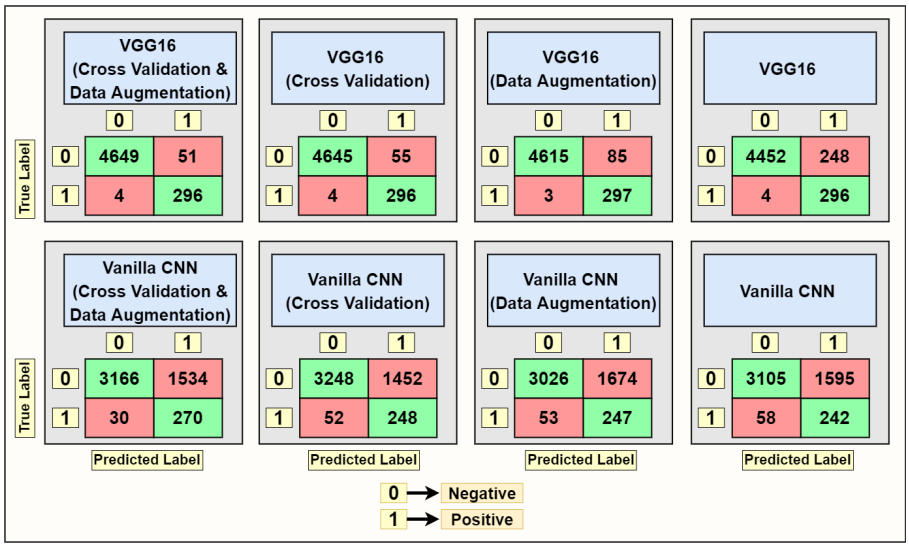

Figure 7.2: Confusion matrices

As we can see in the Roc curve plots of deep learning models in Figure 7.3, the AUC area of pre-trained VGG16 is quite large compared to the AUC area of vanilla CNN. This means that the predictions made with VGG16 have a better probability of predicting true positive and true negative. Our outcomes show us that the application of transfer learning can bring a remarkable improvement in our prediction results. Again, if we look at the AUC areas for VGG16, we can say that the real-time data augmentation implementation increases the AUC area compared to the unimplemented version. While we can clearly see the contribution of cross-validation to the model in confusion matrixes, we cannot see a clear distinction in ROC curves. This shows that using more than one evolution metric allows us to evaluate our models in a better way. For vanilla CNN, we found that among the cross-validated trained versions, the untreated version of real-time augmentation had a slightly larger AUC area. However, we observed that among the versions without cross-validation, the AUC area of the augmented version was larger. We can see that the cross-validated versions have a larger AUC area than the non-applied versions. Contrary to the VGG16 ROC curve results, by examining the vanilla CNN ROC results, we can observe that cross-validation obviously improves the results.

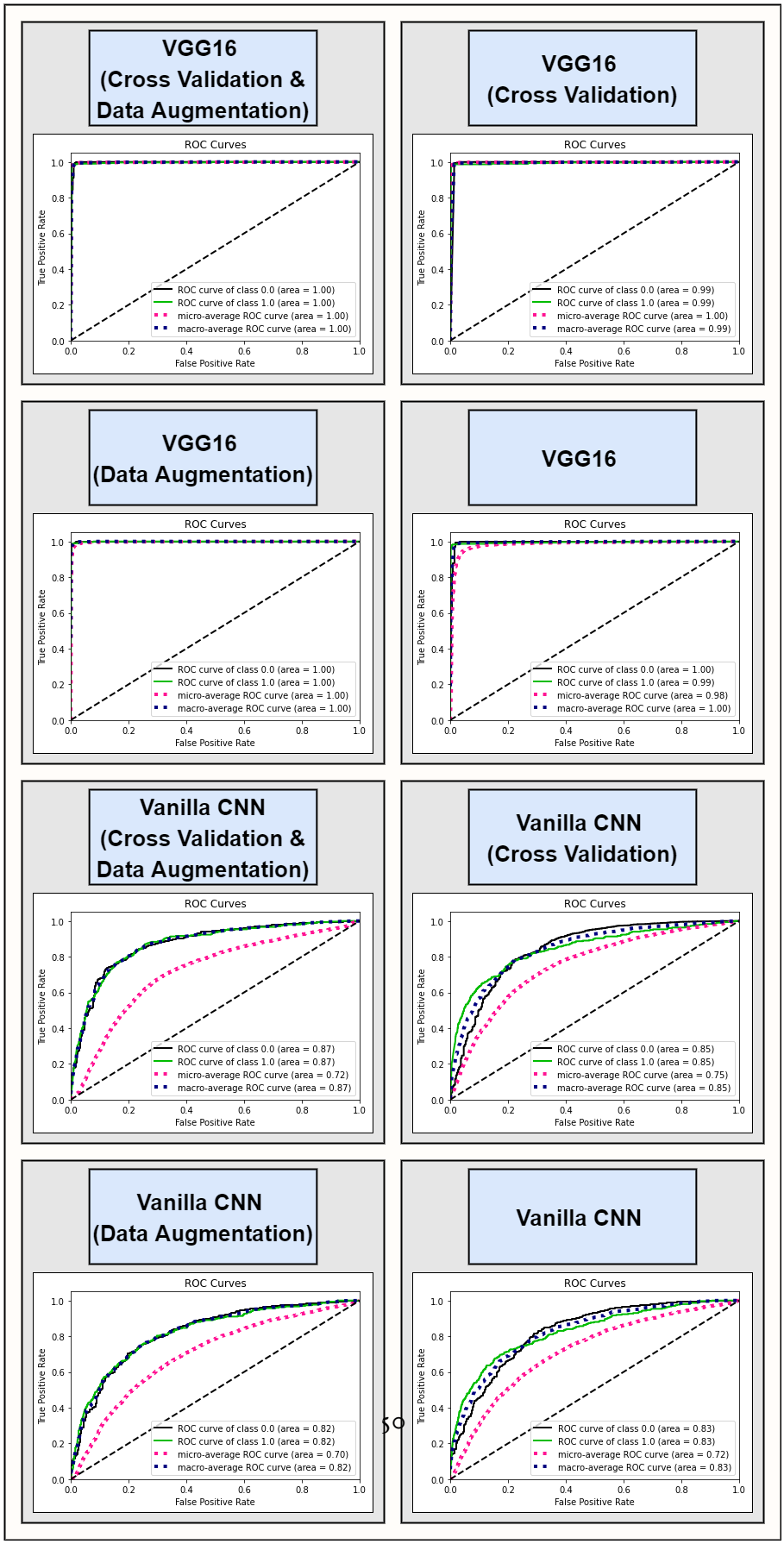

Figure 7.3: ROC curve results of our deep learning models

In addition to the neural network models, SVM and Logistic regression results can be seen in Table 7.2. These two algorithms are trained with and without cross-validation. As can be seen from the results, although applying cross-validation slightly improved the results of Logistic Regression, sufficient results could not be obtained. We can see that, in SVM, applying cross validation did add any improvements to the results.

Table 7.2: Results of machine learning models

Method	Cross Validation	Val Accuracy (avg for folds)	Test Accuracy (avg for folds)	Model Size (KB)
Logistic Reg.	Yes	0,4857	0,5052	393
Logistic Reg.	No	0,4704	0,4822	393
SVM	Yes	0,4647	0,4854	1.439.890
SVM	No	0,4647	0,4854	1.439.890

The ROC curve results of logistic regression in Figure 7.4 show us that this algorithm does not give sufficient results for our problem. Although cross-validation slightly improves the results, the AUC is still very close to 50%. This shows us that the model predicts the classes randomly, which means there is not enough learning. If we observe the results of SVM, even if cross-validation is applied, sufficient learning has not occurred. As in logistic regression, the AUC is very close to 50%.

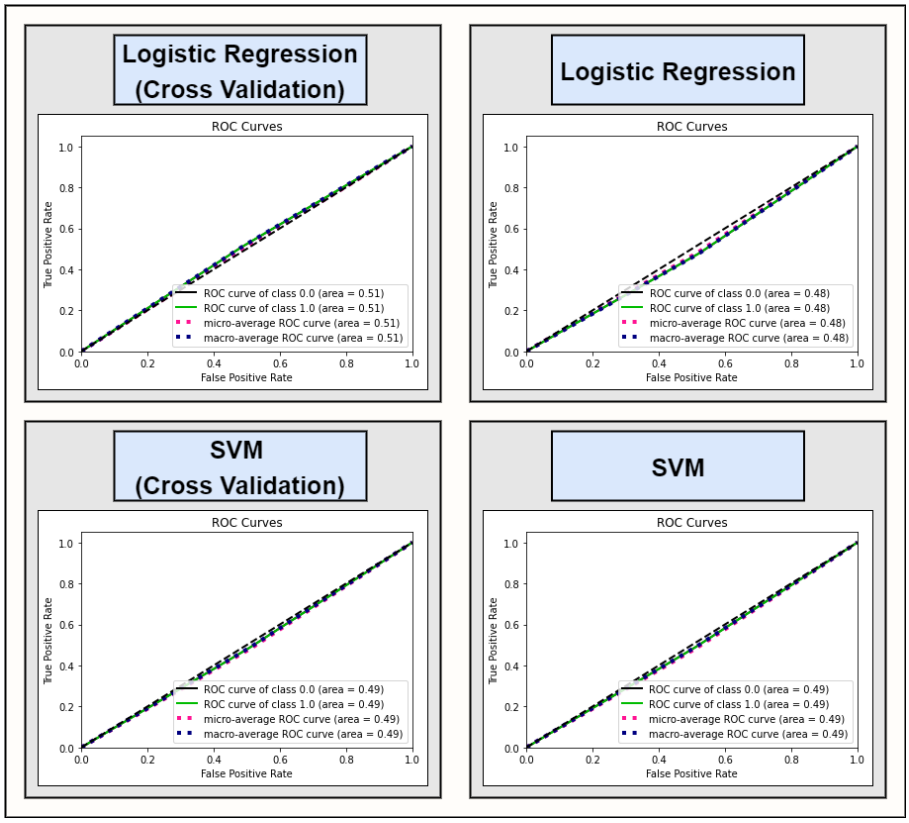

Figure 7.4: ROC curve results for SVM and Logistic regression

8 Conclusion

The primary communication way of the dolphins and an integral part of their existence are whistles. By monitoring and analyzing their sounds, researchers want to secure their populations and gain a better understanding of their habits. Due to the importance of this situation, in this research, a deep learning based dolphin whistle detection framework is proposed for accurately detecting dolphin whistles. The complete framework consists of creating spectrogram images from audios that are recorded underwater with the proposed method, applying preprocessing to them, and detecting the whistles in certain windows with the corresponding models we trained/fine-tuned. Along with the dolphin whistle detection framework, this research contains a comprehensive comparison of the effectiveness of using certain methods during training (data augmentation and cross-validation) together with the comparison of deep learning models, both fine-tuned (VGG16) and trained from scratch (vanilla CNN), against more traditional machine learning algorithms (Logistic Regression, SVM).

Our experimental results show that deep learning models are more effective compared to traditional machine learning algorithms in this task, and that using cross-validation together with data augmentation is a good strategy to mitigate overfitting and improve model performance. The best fine-tuned VGG16 model achieved a 98.9% accuracy on the test set.

In the future, we think that the proposed framework could be adapted for the study of other underwater mammal species and their different type of sounds. Moreover, the number of data collected due to hardware and time limitations was 4198 patterns for training and 5000 for testing. Our study focused solely on the detection of dolphin whistles, and our dataset was pre-

pared accordingly. But besides dolphin whistles, there are other types of dolphin sounds, such as echolocation clicks and burst pulses. Our model can only detect dolphin whistles, however by collecting more samples our method might be adapted to different classification problems, for example to identify other types of dolphin sounds. Also, we are planning to expand the scope of our work with the help of marine biologists who work on marine mammal species, in order to examine which dolphin sounds belong to which dolphin species. In this direction, our goal for our future work is to collect more samples from recordings taken from different regions and to develop our model in a way that detects and classifies different dolphin sounds. In addition, it is aimed to increase the accuracy of our model by increasing the number of samples belonging to each dolphin sound type which means obtaining a larger dataset. Furthermore, besides the models we used in our research, we aim to benchmark our approach against other popular algorithmic methods used for dolphin whistle detection, such as PAMGuard [38].

References

[1] J. Wu, M. Khishe, M. Mohammadi, S. H. T. Karim, and M. Shams, "Acoustic detection and recognition of dolphins using swarm intelligence neural networks," *Applied Ocean Research*, vol. 115, p. 102837, 2021.

[2] D. Kohlsdorf, D. Herzing, and T. Starner, "An auto encoder for audio dolphin communication," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[3] T. Lu, B. Han, and F. Yu, "Detection and classification of marine mammal sounds using alexnet with transfer learning," *Ecological Informatics*, vol. 62, p. 101277, 2021.

[4] C. E. Perazio and S. A. Kuczaj II, "Vocalizations produced by bottlenose dolphins (tursiops truncatus) during mouth actions in aggressive and non-aggressive contexts," *International Journal of Comparative Psychology*, vol. 30, 2017.

[5] P. L. Tyack, "Dolphins whistle a signature tune," *Science*, vol. 289, no. 5483, pp. 1310–1311, 2000.

[6] A. Acevedo-Gutiérrez and S. C. Stienessen, "Bottlenose dolphins (tursiops truncatus) increase number of whistles when feeding," *Aquatic Mammals*, vol. 30, no. 3, pp. 357–362, 2004.

[7] M. Esfahanian, H. Zhuang, and N. Erdol, "On contour-based classification of dolphin whistles by type," *Applied Acoustics*, vol. 76, pp. 274–279, 2014.

[8] H. C. Esch, L. S. Sayigh, J. E. Blum, and R. S. Wells, "Whistles as potential indicators of stress in bottlenose dolphins (tursiops truncatus)," *Journal of Mammalogy*, vol. 90, no. 3, pp. 638–650, 2009.

[9] A. Gannier, S. Fuchs, A. Gannier, M. Fernandez, and J. M. Azevedo, "Dolphin whistle repertoires around são miguel (azores): Are you common or spotted?" *Applied Acoustics*, vol. 161, p. 107169, 2020.

[10] K. M. Dudzinski, "Contact behavior and signal exchange in atlantic spotted dolphins (stenella fimntalis)," *Aquatic Mammals*, vol. 24, pp. 129–142, 1998.

[11] D. L. Herzing, "Vocalizations and associated underwater behavior of free-ranging atlantic spotted dolphins, stenella frontalis and bottlenose dolphins, tursiops truncatus," *Aquatic Mammals*, vol. 22, pp. 61–80, 1996.

[12] V. M. Janik and L. S. Sayigh, "Communication in bottlenose dolphins: 50 years of signature whistle research," *Journal of Comparative Physiology A*, vol. 199, no. 6, pp. 479–489, 2013.

[13] L. J. Miller, M. Solangi, and S. A. Kuczaj II, "Seasonal and diurnal patterns of behavior exhibited by atlantic bottlenose dolphins (tursiops truncatus) in the mississippi sound," *Ethology*, vol. 116, no. 12, pp. 1127–1137, 2010.

[14] M. O. Lammers, W. W. Au, and D. L. Herzing, "The broadband social acoustic signaling behavior of spinner and spotted dolphins," *The Journal of the Acoustical Society of America*, vol. 114, no. 3, pp. 1629–1639, 2003.

[15] M. Thomas, B. Martin, K. Kowarski, B. Gaudet, and S. Matwin, "An end-to-end approach for true detection of low frequency marine mammal vocalizations," *The Journal of the Acoustical Society of America*, vol. 146, no. 4, pp. 2959–2959, 2019.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[18] O. Serra, F. Martins, and L. R. Padovese, "Active contour-based detection of estuarine dolphin whistles in spectrogram images," *Ecological Informatics*, vol. 55, p. 101036, 2020.

[19] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[20] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[22] C. Buchanan, Y. Bi, B. Xue, R. Vennell, S. Childerhouse, M. K. Pine, D. Briscoe, and M. Zhang, "Deep convolutional neural networks for detecting dolphin echolocation clicks," in *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE, 2021, pp. 1–6.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] Y. Shiu, K. Palmer, M. A. Roch, E. Fleishman, X. Liu, E.-M. Nosal, T. Helble, D. Cholewiak, D. Gillespie, and H. Klinck, "Deep neural networks for automated detection of marine mammal species," *Scientific reports*, vol. 10, no. 1, pp. 1–12, 2020.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[26] C. Bergler, H. Schröter, R. X. Cheng, V. Barth, M. Weber, E. Nöth, H. Hofer, and A. Maier, "Orca-spot: An automatic killer whale sound detection toolkit using deep learning," *Scientific reports*, vol. 9, no. 1, pp. 1–17, 2019.

[27] C. Jin, M. Kim, S. Jang, and D.-G. Paeng, "Semantic segmentation-based whistle extraction of indo-pacific bottlenose dolphin residing at the coast of jeju island," *Ecological Indicators*, vol. 137, p. 108792, 2022.

[28] J.-j. Jiang, L.-r. Bu, F.-j. Duan, X.-q. Wang, W. Liu, Z.-b. Sun, and C.-y. Li, "Whistle detection and classification for whales based on convolutional neural networks," *Applied Acoustics*, vol. 150, pp. 169–178, 2019.

[29] A. K. Ibrahim, H. Zhuang, N. Erdol, and A. M. Ali, "A new approach for north atlantic right whale upcall detection," in *2016 International Symposium on Computer, Consumer and Control (IS3C)*. IEEE, 2016, pp. 260–263.

[30] P. Li, X. Liu, K. Palmer, E. Fleishman, D. Gillespie, E.-M. Nosal, Y. Shiu, H. Klinck, D. Cholewiak, T. Helble *et al.*, "Learning deep models from synthetic data for extracting dolphin whistle contours," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–10.

[31] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[32] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[35] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," *Advances in neural information processing systems*, vol. 2, 1989.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[37] J. W. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, pp. 99–114, 1949.

[38] D. Gillespie, M. Caillat, J. Gordon, and P. White, "Automatic detection and classification of odontocete whistles," *The Journal of the Acoustical Society of America*, vol. 134, no. 3, pp. 2427–2437, 2013.

Acknowledgments

I would like to reveal my special thanks of gratitude to my supervisor as well as my co-supervisor, who gave me the opportunity to do this great project on the topic of "Deep learning techniques for biological signal processing: Automatic detection of dolphin sounds"which also helped me in doing much research and improved my skills a lot during the process. I am thankful to them.