Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning

Thesis by

Nura Aljaafari

In Partial Fulfillment of the Requirements

For the Degree of

Masters of Science

King Abdullah University of Science and Technology

Thuwal, Kingdom of Saudi Arabia

(February, 2018)
The thesis of Nura Aljaafari is approved by the examination committee.

Committee Chairperson: Panagiotis Kalnis, Professor
Committee Members: Carlos M. Duarte, Professor
Committee Members: Xin Gao, Associate Professor
ABSTRACT

Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning

Nura Aljaafari

The study and the analysis of marine ecosystems is a significant part of the marine science research. These systems are valuable resources for fisheries, improving water quality and can even be used in drugs production. The investigation of ichthyoplankton inhabiting these ecosystems is also an important research field. Ichthyoplankton are fish in their early stages of life. In this stage, the fish have relatively similar shape and are small in size. The currently used way of identifying them is not optimal. Marine scientists typically study such organism by sending a team that collects samples from the sea which is then taken to the lab for further investigation. These samples need to be studied by an expert and usually end needing a DNA sequencing. This method is time-consuming and requires a high level of experience. The recent advances in AI have helped to solve and automate several difficult tasks which motivated us to develop a classification tool for ichthyoplankton. We show that using machine learning techniques, such as generative adversarial networks combined with transfer learning solves such a problem with high accuracy. We show that using traditional machine learning algorithms fails to solve it. We also give a general framework for creating a classification tool when the dataset used for training is a limited dataset. We aim to build a user-friendly tool that can be used by any user for the classification task and we aim to give a guide to the researchers so that they can follow in creating a classification tool.
ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Professor Panos Kalnis for his patience, guidance, and support that he has provided throughout my time as his student. His guidance helped me to be a better researcher.

I am thankful to all members of the InfoCloud group for the help and support they provided. To my KAUSTian friends, thank you for making my experience at KAUST indelible. My sincere thanks also goes to my family and my friends for their endless support.
# TABLE OF CONTENTS

- Examination Committee Page ........................................... 2
- Copyright ........................................................................... 3
- Abstract ............................................................................ 4
- Acknowledgements ............................................................ 5
- List of Figures ..................................................................... 9
- List of Tables ....................................................................... 10

## 1 Introduction .................................................................. 11

## 2 Related Work ............................................................... 14
### 2.1 Support Vector Machines .............................................. 14
### 2.2 Short-time Fourier transform and discriminant analysis .......... 16
### 2.3 Artificial Neural Networks (ANNs) .................................. 16
### 2.4 Classification tools built with other algorithms ..................... 17
### 2.5 Commercial Applications ............................................... 19

## 3 Methodology ................................................................. 20
### 3.1 Dataset ........................................................................ 20
### 3.2 Training Procedure ....................................................... 23
### 3.3 Learning Algorithms Classes ......................................... 25
#### 3.3.1 Learning algorithms families ..................................... 25
### 3.4 Standard classifiers ....................................................... 26
### 3.5 Data Augmentation ......................................................... 30
### 3.6 Deep Learning .............................................................. 31
#### 3.6.1 Training a Convolution Neural Network from Scratch ........ 37
#### 3.6.2 Transfer Learning ...................................................... 38
#### 3.6.3 Generative Adversarial Networks (GANs) ....................... 40
#### 3.6.4 Transfer Learning with Generative adversarial networks (GANs) 46
4 Experimental Evaluation

4.1 Experiment setup .............................................. 50
4.2 Results .......................................................... 50
  4.2.1 Standard classifiers ....................................... 50
  4.2.2 Training ConvNet from scratch ......................... 52
  4.2.3 Transfer learning ......................................... 53
  4.2.4 Semi-supervised learning with generative adversarial network (SSGAN) ........................................... 55
  4.2.5 Semi-supervised learning with generative adversarial network (SSGAN) and transfer learning ......................... 56

5 Concluding Remarks .............................................. 57
  5.1 Future Research Work ....................................... 57

References .......................................................... 58
LIST OF FIGURES

3.1 Sample of the Fish Dataset ........................................ 21
3.2 Data Distribution of the Families in the Dataset .................. 22
3.3 Example of the Data Augmentation Operations ..................... 30
3.4 Example of convolution operation ................................... 33
3.5 Example of pooling operations ...................................... 34
3.6 Dropout operation ...................................................... 36
3.7 The ConvNet architecture used in training a network from scratch . 38
3.8 Example of how transfer learning works ............................ 40
3.9 Vanilla GAN on the left and the updated GAN on the right ........ 42
3.10 The discriminator architecture used in the GAN model .......... 45
3.11 The generator architecture used in the GAN model .............. 46
3.12 Modified version of VGG19 used as the discriminator architecture 47
3.13 Sample of GAN Synthetic Data ..................................... 47
3.14 The tool framework .................................................... 48
3.15 Images representation throughout the network .................... 49
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Standard Classifiers Results for the Top 10 Families</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Standard Classifiers Results for the Top 20 Families</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Standard Classifiers Results for the Top 31 Families</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Standard Classifiers Results for the Top 42 Families</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>ConvNet Results for the Top 10 Families</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>ConvNet Results for the Top 20 Families</td>
<td>53</td>
</tr>
<tr>
<td>4.7</td>
<td>ConvNet Results for the Top 31 Families</td>
<td>53</td>
</tr>
<tr>
<td>4.8</td>
<td>ConvNet Results for the Top 42 Families</td>
<td>53</td>
</tr>
<tr>
<td>4.9</td>
<td>Transfer Learning Results for the Top 10 Families</td>
<td>54</td>
</tr>
<tr>
<td>4.10</td>
<td>Transfer Learning Results for the Top 20 Families</td>
<td>54</td>
</tr>
<tr>
<td>4.11</td>
<td>Transfer Learning Results for the Top 31 Families</td>
<td>54</td>
</tr>
<tr>
<td>4.12</td>
<td>Transfer Learning Results for the Top 42 Families</td>
<td>54</td>
</tr>
<tr>
<td>4.13</td>
<td>SSGAN Results for the Top 10 Families</td>
<td>55</td>
</tr>
<tr>
<td>4.14</td>
<td>SSGAN Results for the Top 20 Families</td>
<td>55</td>
</tr>
<tr>
<td>4.15</td>
<td>SSGAN Results for the Top 31 Families</td>
<td>55</td>
</tr>
<tr>
<td>4.16</td>
<td>SSGAN Results for the Top 42 Families</td>
<td>55</td>
</tr>
<tr>
<td>4.17</td>
<td>SSGAN with Transfer Learning Results for the Top 10 Families</td>
<td>56</td>
</tr>
<tr>
<td>4.18</td>
<td>SSGAN with Transfer Learning Results for the Top 20 Families</td>
<td>56</td>
</tr>
<tr>
<td>4.19</td>
<td>SSGAN with Transfer Learning Results for the Top 31 Families</td>
<td>56</td>
</tr>
<tr>
<td>4.20</td>
<td>SSGAN with Transfer Learning Results for the Top 42 Families</td>
<td>56</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Artificial intelligence (AI) is playing a significant role in shifting the progress of many real-world applications [1, 2, 3]. Self-driving cars, automating the production in factories, and virtual agents are some of the examples of this shift. One important field of AI is deep learning which is believed to be the core of the recent advances in AI. Deep learning techniques are machine learning algorithms that are characterized as representation learning algorithms, meaning, they have the ability to learn and extract the data features without explicit feature engineering from the user [3, 4]. Deep learning has helped to improve the state-of-the-art responses to several challenging problems such as image classification, speech recognition, and object detection [5, 6, 7, 8, 9]. These advances have motivated us to look for a problem where AI can be used to solve it more efficiently and more easily. The study and analysis of marine ecosystems is an interesting problem that can be improved and made more accessible to scientists via AI.

Marine waters cover two-thirds of the surface of the Earth [10], oceans and seas across the world have great diversity in their ecosystems. Marine ecosystems provide valuable services and goods [11] including fisheries production, carbon sequestration, and improving water quality. Furthermore, discovering marine ecosystems may help to explore potential medicines and developing new drugs [12]. It can also be useful in understating challenges resulting from the impact of humans and global warming on the ecosystems and how to address them. Exploring and understating these marine ecosystems have been a significant research interest of the marine scientists [13].
A similar study has been conducted by the Red Sea Research Center (RSRC) at King Abdullah University of Science and Technology (KAUST). One major part of their research is to document the oceanographic and environmental gradients in the Red Sea based on the existing literature, data and further investigation. As part of their research, the marine scientists are studying the ichthyoplankton inhabiting the Red Sea. Ichthyoplankton is a term used for fish eggs and larvae that are found near the near-surface waters [14]. It is one of the early stages of a fish life cycle. At this stage, the fish are small in size, and they usually need to be examined using microscopes to identify their species and families.

The typical method for studying ichthyoplankton is by sending a team to collect samples which are then taken to the lab for further investigation. The samples are examined by an expert to identify the different organisms that the samples belong to. This process typically ends up with a need for a DNA sequencing, especially, when dealing with small or microscopic organisms that are hard to identify due to the significant similarities among different species. This approach is not optimal due to many reasons; it is time-consuming, labor-intensive and requires an expert. Automating the process could help in resolving several issues; if a system was created for fish recognition, then the process can be carried out by any marine scientist or user without the need of an expert in the field to be present.

To the best of our knowledge, there has not been any proposed system for the identification of the Red Sea fish. Also, all the proposed solutions in the literature for other fish are focusing on the recognition of the large and full-grown fish without a designated system for the recognition of the fish in the early stage of their life cycle, specifically when they are ichthyoplankton.

Thus, we proposed a user-friendly tool that can be used by any marine scientist or user without any required level of expertise that is able to recognize which family a fish belongs to, given its image.
We aim to give a tool that needs no expertise on machine learning to be used for the Red Sea ichthyoplankton. We also aim to give a framework for the researchers to follow when designing a classification tool for a limited dataset.

The tool was developed and trained on a fish dataset provided by the RSRC at KAUST. It was a dataset of labeled images for ichthyoplankton of the Red Sea. The dataset has a challenging problem that it is limited in the number of samples each class contains. Several methods where used to run the tests, we have examined with developing the tool using standard classifiers of machine learning such as support vector machines and decision trees. We have also tested with training a neural network from scratch, transfer learning from a pre-trained network into a new network that we used to classify our dataset. Furthermore, we tried modifying a generative adversarial network (GAN) to work as a classifier. All the mentioned methods failed to solve the classification problem. None of them has even reached a 46% level of accuracy. When combining transfer learning with a GAN, we achieved a 72% accuracy. Using this method, we also reached a significant improvement in the precision and the recall score.

The thesis is divided as follows; chapter one gives an overview of the problem we are trying to solve, chapter two describes some of the related work and the available tools for solving similar problems. In chapter three we give a detailed explanation of all the stages that the development has been through along with a detailed description of the used dataset and the training procedure. Chapter four describes the result we got from every stage. Chapter five presents a summary of the thesis and closing remarks.
Chapter 2

Related Work

To the best of our knowledge, there is no system for classification of the Red Sea ichthyoplankton. We list here some of the available systems for fish classification based on their images. All of the available systems were designed to classify full-grown fish, and all were designed in a supervised learning approach. In contrast to our dataset, the datasets these systems used were of fish that are easily distinguished by the naked eye. We list some of the systems designed in the academia and some of the systems that are available for commercial use.

Most of the solutions were built based on machine learning algorithms such as Support Vector Machines (SVMs), and Artificial Neural Networks (ANNs). We list below these tools categorized based on their core algorithm alongside with some commercial tools.

2.1 Support Vector Machines

A Support vector machine (SVM) is arguably one of the best supervised learning algorithms and one of the most commonly used ones. It is a discriminate algorithm that aims to find a separation line in the hyperplane between two or more classes subject to maximizing the distance (margin) between the line and the points of the classes. Due to the nature of convex optimization, SVM classifiers are guaranteed to find the optimal solution. They are useful for linearly separable cases and non-linear separable cases.
In [11], the authors propose a binary classifier to two species of fish, namely, the Ethmalosa fimbrita and the Scomberomorous tritor. The authors considered five-length features, specifically, the anal, the caudal, the dorsal, the pelvic and the pectoral. The choice of these features was due to their commonality in classifying fish. The data used was labeled, and it consisted of 150 images per species where 76 images were used for training, and 74 images were used for testing. The proposed classifier was built using the SVM method and achieved an approximate accuracy of 74.32% which was significantly higher than the accuracy obtained from building a classifier using an artificial neural network (ANN), k-nearest neighbor (KNN) and K-mean clustering-based algorithm for the same dataset. The model presented obtained reasonable accuracy, but it is limited. It is a binary classifier tested on a small dataset and the species chosen are relatively easy to distinguish.

The authors of [15], present a classification method of six different species of fish using multi-class SVM (MSVM). They proposed a model that uses images taken by mobile phones by farmers in rural areas to classify fish into their species which will help the farmers later in identifying sick fish without the need of an expert present. The images are pre-processed by cropping the original image to a new image that contains the fish’s skin without any other details of the fish shape such as the length or the width. For the feature extraction, first, HSV color features were added to RGB color features which combined will be used as the color features. Then two different types of statistical texture features were extracted; namely, a grayscale histogram (GH) based texture and a gray level co-occurrence matrices (GLCMs) based texture. Finally, norm entropy and norm energy for decomposed images of fish skin were extracted as a wavelet-based texture feature. For feature selection, LINSVM was used to select the most accurate feature group vector from the features mentioned earlier. For the classification, a one-against-one (OAO) based MSVM was constructed. The dataset they used consisted of 540 fish images, all labeled. For the testing, leave-one-
out cross-validation was used. On average the model obtained an accuracy of 95.92%. Although this model achieved high accuracy, the data that was used is relatively easy to be classified. The different fish have different skins that highly vary in their shape, which played an essential role in reaching such accuracy. If, however the model had been tested on images of fish with similar skin type, we suspect that it would not reach such accuracy. As shown in section 4, we have experimented with building a classification tool using SVMs, but the models built using SVM fails to solve our classification problem.

2.2 Short-time Fourier transform and discriminant analysis

There are different characteristics which can be used to classify fish, and one interesting feature is the otolith contour. Otoliths are “hard, calcium carbonate structures located directly behind the brain of bony fishes [16]”. The authors here [17], built a classification model of fish that can classify fourteen species of fish with an approximate accuracy of 92% based on their otolith contour images. First, the model extracts the features and analyzes the shape of the contour using Short-time Fourier transform (STFT). Then the results are utilized as inputs to a Discriminant Analysis classifier (DA) for classification. Their dataset has 392 otolith contour images where each species has 28 images spanning three families; Sciaenidae, Ariidae, and Engraulidae. We could not test such method on our dataset due to the lack of the otolith contour images.

2.3 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are non-linear, statistical, data-modeling tools used for modeling complex relationships between inputs and outputs or finding pat-
terns in data [18]. ANNs were widely employed in the literature as a classification algorithm as in [19, 20] but these classifiers were trained to classify the fish based on echo recordings rather than their images. They were also used as in [21] where a complicated task of taking the fish images is needed (following the approach of [22]) followed by another complicated task for the images processing with the need of three VME-bus based printed circuit boards. Finally, the images were classified using a simple ANN. This method proposes a very complicated task for the image taking, processing, and classification which defies the intention of our method, building a user-friendly tool that requires no particular expertise in machine learning or software engineering.

On a more recent paper [23], ANN was used as an image classifier where each image is classified after a series of feature engineering. Each image was processed to extract 15 different features divided into geometrical, morphological, and texture features. These features are then passed to a neural network with a single hidden layer with sigmoid as its activation function.

2.4 Classification tools built with other algorithms

In [21], images were taken for a group of 5 fish; megrim, whiting, saithe, haddock, gurnard, and herring. These images were taken with a white background, and 50 points were measured, then these 50 points were used to extract three attributes: invariant moments, minimum mismatch factor and shape descriptors. They measured the differences between two images by a mismatch factor where one image is considered as a template, and the other one as a testing point; they call this classification by optimization. The last method was called recognition using geometric shape descriptors, where 11 shape descriptors were calculated for each fish (the shape descriptors) then these values were used to classify the fish into the different species.
This method is very restrictive since it constrains the image of the fish to be taken with a white background and the preprocessing step with the required measurements is not feasible for all types of fish.

The system proposed in [24] is one of the few systems available for ichthyoplankton recognition, but it was built as an aiding system to the existing imaging machine, namely Situ. They focused on studying five taxa of interest, namely Trichodesmium, Larvaceans, Fish larvae, Copepods, and Chaetognaths with samples between 202 and 248 per class. They start preprocessing each image by removing the background using a gray-scale intensity distribution of the expected background which is expected to leave only the organism in the image. Then for the classification task, they use the Scale Invariant Feature Transform (SIFT) which matches and compares the detected organism with the stored images in their dataset. They reached on average a recognition rate of 76.4%.

The methods mentioned in the previous sections perform well when there is a sufficient training data, and the fish features can be disguised easily. Although they were shown to achieve a relatively high accuracy, they were trained on a limited number of species that can be easily classified. They also require feature engineering, which can be extremely challenging for fish in early stages as ichthyoplankton and would also require some level of expertise in computer science for the user. Furthermore, the proposed solutions can be considered as out-dated. Given that the state-of-the-art in image recognition is methods that are built using deep learning, none of the proposed solutions implemented deep learning as part of their systems. Another downside of these methods is that they assume that the images have to be taken in a special way as well as positioning the fish.
2.5 Commercial Applications

Commercially, there were several attempts to build fish classifiers such as FishVerfy [25], FishyID [26], FishApp Atlas [27], and WhatsThatFish [28]. These applications vary from being like an image dictionary where each fish image has its name as in FishApp Atlas, and FishyID. In other applications such as FishVerfy, and WhatsThatFish the user takes a photo of the fish then the application identifies it. FishVerfy is the most recent application, and it uses deep learning and image recognition techniques, it, however, does not support the identification of the same species that we have in our dataset. It also comes only in as a mobile application without the support of being used in desktop computers. FishID does not identify fish by their images but given a geolocation; it lists the known fish in this location, then the user can compare the fish they have with the available data. FishApp Atlas works as an encyclopedia, it does not identify fish, but it provides information about several fish families. WhatsThatFish is a website design for the identification of tropical fish. It was built as a collaborative community where the users submit the data they have and can view data provided by other users.

All the listed software above are specialized in the identification of grown fish not when they are in the ichthyoplankton stage, and none of them was designed to be used for the identification of the Red Sea’s fish or ichthyoplankton.
Chapter 3

Methodology

We created a framework that any user or researcher can use to benchmark and deal with limited datasets. We focused on creating a simple tool that a user needs only to give an image then the image is going to be classified. If the user wanted to add more data to enhance the performance all they need to do is to give a path to the directory of images without any extra work.

In this chapter we discuss the following, we give an overview in 3.1 about the dataset used in this thesis. Section 3.2 explains the training process, how the dataset was divided, what are the training algorithms used, and what are the metrics used for evaluating the performance. Section 3.3 presents some general terms used in the thesis. The framework is shown in several sections starting by 3.4 with traditional machine learning algorithms then 3.5 for the data augmentation. Deep learning methods and techniques are explained in details in section 3.6.

3.1 Dataset

The dataset of the project was supplied by the Red Sea Research Center (RSRC) at KAUST. It is a collection of Ichthyoplankton images that were collected on a monthly basis in 2013 (from January to December) near Thuwal, Saudi Arabia. Two distinct sites in close proximity to coral reefs were sampled once per month: one shallow inshore station (ca. 50 m depth) and a nearby offshore, deeper station (~
500 m depth). The samples (larvae) were preserved in absolute ethanol to allow molecular analysis. For the image taking, Leica MDG33 Stereoscope equipped with a Leica Z6 APO zoom system, a digital camera LEICA IC80 HD and a computer were used. Different magnifications were used based on the larvae, but in most cases, the magnification was between x10, x12.5, x16 and x20.

We call this dataset the Fish dataset. The Fish dataset has 1699 images that are divided into different 61 families which are divided into 327 species. The families are Acanthouridae, Antenarridae, Apogonidae, Balistidae, Blenniidae, Bothidae, Bregmacerotidae, Caesionidae, Callionymidae, Carangidae, Chanidae, Clupeidae, Congridae, Echeneidae, Engraulidae, Ephippidae, Fistularidae, Gempylidae, Gerreidae, Gobiidae, Haemulidae, Holocentridae, Hoplichthyidae, Labridae, Leiognathidae, Lethrinidae, Lutjanidae, Microdesmidae, Monacanthidae, Monodactylidae, Mullidae, Myctophidae, Nemipteridae, Nomeidae, Ophididdae, Paralepididae, Pempheridae, Phosichthyidae, Pinguipedidae, Platycephalidae, Pomacanthidae, Pomacentridae, Priacanthida, Pseudochromidae, Scaridae, Schindleriidae, Scombridae, Scorpaenidae, Serranidae, Siganidae, Soleidae, Sphyraenidae, Sternoptychidae, Stomiidae, Syngnathidae, Synodontidae, Tetraodontidae, Triacanthidae, Trichiuridae, Tripterygiidae, and Zeidae.

The dataset is highly imbalance wherein the extreme case we have a single sample for one family and 402 samples for one family. Figure 3.2 shows the data distribution over the families in the dataset.
Figure 3.2: Data Distribution of the Families in the Dataset

The given dataset needed several preprocessing steps. Some of the images contained multiple fish in a single image or noisy parts in it such as the forceps and so on. Those images were processed so that each image contains a single fish. A sample of the dataset is shown in figure 3.1.

We divided the dataset into four categories, Top 10, Top 20, Top 31, and Top 42. The Top 10 category has ten families with the highest number of samples, the Top 20 has 20 families with the highest number of samples, and similarly, Top 31 and Top 42 were created. The remaining families were excluded from the training due to the insufficient number of samples that they have. We omit dividing the data according to their species for two main reasons. First, the number of samples per species is limited. Second, even though the data was labeled by an expert, there is a good probability that the samples were labeled with an incorrect species due to the high similarities among the species.

Top 10 has the following families, Apogonidae, Blenniidae, Caesionidae, Carangidae, Gobiidae, Labridae, Pomacentridae, Scaridae, Scombridae and Serranidae. For the Top 20 families, it has the Top 10 families in addition to the following families, Bothidae, Bregmacerotidae, Clupeidae, Lethrinidae, Microdesmidae, Myctophidae,
Nemipteridae, Phosichthyidae, Schindleridae and Stomiidae. The following families were added to the Top 20 families to create the Top 31 families, Balistidae, Gerreidae, Holocentridae, Leiognathidae, Monacanthidae, Paralepididae, Phosichthyidae, Pomacanthidae, Priacanthidae, Sphyraenidae, Synodontidae and Trichiuridae. Finally, to create the Top 42 families, these families were added, Balistidae, Callionymidae, Congridae, Engraulidae, Ephippidae, Gerreidae, Holocentridae, Leiognathidae, Monacanthidae, Nomeidae, Paralepididae, Pempheridae, Phosichthyidae, Pomacanthidae, Priacanthidae, Pseudochromidae, Scorpaenidae, Siganidae, Sphyraenidae, Sternopterygidae, Synodontidae, Tetraodontidae and Trichiuridae. The dataset was also augmented using several methods that are discussed in details in section 3.5.

### 3.2 Training Procedure

Following the common practice in machine learning, the dataset was divided into three sets; a training set, a validation set and a testing set with 80% for training, 10% for validation, and 10% for testing. For the testing metrics, accuracy, precision and recall were the metrics chosen with the main focus on achieving high accuracy. Since the problem is a multi-class classification problem, the classical way of calculating those metrics was modified to fit this setting, following the method described in [29], the formulas that are shown in equations (3.1), (3.2) and (3.3) show how the accuracy, the precision and the recall are calculated.

\[
\text{Accuracy} = \frac{\sum_{i=1}^{l} \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i}}{l}
\]  
\[\text{(3.1)}\]

\[
\text{Precision} = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l}
\]  
\[\text{(3.2)}\]
Recall = \frac{\sum_{i=1}^{l} tp_i}{ tp_i + fn_i } \quad (3.3)

where \( tp \) is the true positive, \( tn \) is the true negative, \( fp \) is the false positive, \( fn \) is the false negative, and \( l \) is the number of classes.

The micro-precision, and micro-recall (shown in 3.4 and 3.5) were also measured to evaluate the different models’ performances.

\[ \text{Precision}_{\text{micro}} = \frac{\sum_{i=1}^{k} tp_i}{ \sum_{i=1}^{k} (tp_i + fp_i) } \quad (3.4) \]

\[ \text{Recall}_{\text{micro}} = \frac{\sum_{i=1}^{k} tp_i}{ \sum_{i=1}^{k} (tp_i + fn_i) } \quad (3.5) \]

where \( k \) is the number of classes.

The provided dataset was never used as a training set for machine learning; hence, it was never benchmarked. We benchmarked the dataset using the standard classifiers (see section 3.4 for further discussion of mentioned classifiers).

The standard classifiers were trained using the K-Folds cross-validation method, with \( K \) equals to ten. K-Folds cross-validation is a special case of the cross-validation method where the training set is partitioned into \( k \) equally (or nearly equally) sized segments or folds [30]. Then the classifier is trained on \( K \) training iterations such that in each iteration 1 set is reserved for validation and the other \( K - 1 \) sets are used for training. Upon testing with different values of \( K \), ten showed the best results which follows from the suggested method in [31] and as stated earlier 10% of the data was kept as the testing set, and the remaining 90% of the data was used for the training.

For the deep learning methods, they were trained using back-propagation algorithm and stochastic gradient descent with the cross-entropy as the cost function. Further details for this algorithm are shown in 3.6.
3.3 Learning Algorithms Classes

We define here some terms and terminologies that are used in the following sections.

3.3.1 Learning algorithms families

The learning algorithms are divided into three families, supervised learning, unsupervised learning and semi-supervised learning. Supervised learning algorithms are the family of the algorithms where for each training data point, there is a label associated with it that defines to which class it belongs. To define supervised learning mathematically, let \( X = (x_1, ..., x_n) \) be a set of \( n \) samples (or points), where \( x_i \in X \) for all \( i \in [n] := 1, ..., n \) with the assumption that the points are i.i.d. (independently and identically distributed) from a common distribution on \( X \) and \( Y = (y_1, ..., y_m) \) be a set of \( m \) labels. Let the training points be pairs of \( (x_i, y_i) \) that are sampled i.i.d. from some distribution which here ranges over \( X \times Y \). The goal of supervised learning is to learn the mapping from \( x \) to \( y \).

The unsupervised learning algorithms are the extreme opposite of the supervised learning ones. In this setting the training data points come with no information about their label or class. To define the unsupervised learning algorithms mathematically, let \( X = (x_1, ..., x_n) \) be a set of \( n \) samples (or points), where \( x_i \in X \) for all \( i \in [n] := 1, ..., n \) with the assumption that the points are i.i.d. (independently and identically distributed) from a common distribution on \( X \), the unsupervised-learning algorithms aim to find an interesting structure or pattern in the data \( X \).

A middle ground solution is the semi-supervised learning algorithms. Semi-supervised learning is a specialized class of learning where classifiers use labeled data combined with unlabeled data to build the model. In semi-supervised learning, the data is divided into two parts; \( X_l := (x_1, ..., x_l) \), for which labels \( Y_l := (y_1, ..., y_l) \) are provided, and the points \( X_u := (x_{l+1}, ..., x_{l+u}) \), the labels of which are not known.
Another classification of the learning algorithms is whether they are discriminative or generative learning algorithms. Discriminative learning algorithms are the algorithms that try to identify or model $p(y|x)$, which are the mappings directly from the space of inputs $X$ to the labels $y$. Logistic regression, SVM and random forests are some of the examples of the discriminative learning algorithms. These learning algorithms are known for their good performance, especially, when a large number of labeled data is available \[33\]. However, they are not able to make any use of unlabeled data \[33\].

On the other hand, if the algorithm tries to learn $p(x|y)$ and $p(y)$, it is considered as a generative model, meaning generative models learn how to describe the data distribution using a probabilistic model. Generative models are capable of more than data classification. For example, they can be used for data generation, or imputing missing data. They can also be used in more than a supervised learning, namely semi-supervised learning and unsupervised learning which makes them an excellent candidate for using when the labeled data is scarce.

### 3.4 Standard classifiers

The first attempt to build the classification tool was using standard classification algorithms. These algorithms are relatively cheaper to build and faster to train than using deep learning models. Standard classifiers have shown several successful results in image classification \[34\] \[35\] \[36\]. All the algorithms described in here were trained in a supervised learning manner.

The algorithms were: k-nearest neighbor (KNN), support vector machines (SVMs), decision tree, random forests, multi-layer perceptron, Ada-boost and naive Bayes. Below is a short description of each algorithm.
K-nearest neighbor (KNN) is a lazy learning algorithm, meaning it does not learn a discriminative function from the training data but memorizes the training dataset instead [37]. There are two main parameters in this algorithm: a predefined number K and a distance metric. When a new data point needs classification, it is assigned the label of the nearest examples to it where the distance is measured via the distance metric. In principle, the distance metric can be any metric measure. In our case, K was chosen as 3, and the standard Euclidean distance was used as the distance metric.

Support vector machines (SVMs) is believed to be one of the top ranking algorithms in machine learning [38] that has strong theoretical foundations and wide practical applications [39]. SVMs are supervised and discriminate learning algorithms that aim to arrange the training examples in a separating hyperplane such that margin or the distance between the different groups is maximized. This is achieved by using non-linear kernel functions which map the training data to a high-dimensional feature space while aiming to maximize the margin between the training data in the new feature space. The main parameters are the kernel, $C$ and $\gamma$. The kernel is the similarity function used by the model to determine how similar two points are. $C$ defines how the model handles the misclassification of the data. A low $C$ makes the decision choice smoother, while a large make $C$ forces the model to classify all the training examples correctly which potentially leads to overfitting. $\gamma$ determines the influence of a single training point over the decision boundary. If $\gamma$ has a high value, then the decision boundary is affected by the closest points to it while if $\gamma$ has a low value then even the far points are taken into consideration when updating the decision boundary. By performing a grid search we found that to get the best result for our classifier, we used a radial basis function (RBF) as the kernel, a $\gamma$ equals 2, and a $C$ equals 1.
Decision trees (DTs) are among the oldest and most widely used algorithms in machine learning [38] that are described as non-parametric supervised learning algorithms. The model learns to perform the classification task by creating a decision tree that keeps expanding by splitting based on the features space of the data. The split in each level is chosen by a metric called information gain (IG) where we aim to maximize it. The process is done iteratively until the leaves are clearly divided into the different classes.

Random Forests are considered to be an ensemble of decision trees. The model is built by combining several weak learners (i.e., several DTs) which results in a robust model, with better generalization and less prone to overfitting than DTs [39]. The model is generated by creating a DT from a bootstrap sample. Then $d$ features are randomly selected to be used in calculating IG and splitting the nodes. The process is repeated $k$ times. Then those trees are combined to generate a random forest. The class label for a new data point is chosen by a majority vote from the different trees.

Multilayer perceptron (MLP or Artificial Neural Network - ANN) is a feed-forward artificial neural network that can be used for supervised, unsupervised and semi-supervised learning. MLP is built of several connected layers including an input layer, an output layer and one or more hidden layers. The model runs by combining its input values by a weighted sum that is passed to an activation function that is usually chosen as a sigmoid function or a rectified linear unit (ReLU). Finally, the value is passed to the output layer. Mathematically, this process can be modeled by equation 3.6:

$$o = f\left(\sum_{n=1}^{D} w_n\right)$$  \hspace{1cm} (3.6)

In case of supervised learning, the models built using MLPs are typically trained using the back-propagation algorithm. In the recent years, ANNs have been advancing and improving due to the techniques of deep learning which are discussed in details in 3.6.
Ada-Boost is a machine learning meta-estimator that is used in conjunction with other machine learning algorithms to improve their performance [40]. Ada-Boost works in a similar way to the random forests, it runs in an iterative manner wherein each iteration a new weak learning algorithm is added and the results of the several weak learning algorithms are combined to build a strong learning algorithm. The weighted sum is used as the way it combines the results of the weak classifiers where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases [37].

Gaussian Naïve Bayes (GaussianNB) is a simple and an efficient machine learning method and a probabilistic classifier built on the essence of Bayes theorem. It depends on the conditional probability given by equation 3.7 which predicts the class of input based on the feature vector of it with a prior distribution of the probability [41]. In equation 3.7, \( P(y|x) \) is the posterior probability of class \( y \) given attribute \( x \), \( P(y) \) is the prior probability of class \( y \), \( P(x|y) \) is the probability of attribute \( x \) given class \( y \) (also known as the likelihood), and \( P(x) \) is the prior probability of the attribute \( x \).

\[
P(y | x) = \frac{P(x | y) P(y)}{P(x)}
\]

All the discussed algorithms failed to solve the classification problem when using the original dataset with no augmentation. The performance of the algorithms differs based on the category of the dataset that they are being trained with. In the case of using the Top 10 families, SVM performed best with 32.5 % as its accuracy score. When testing Top 20, SVM, DT, MLP and Ada-boost performed very similarly, with Ada-boost giving the best performance with 28.9 %. For the Top 31 families, SVM outperformed all the other algorithms with the same accuracy as in the previous case, 27.7 %. Finally, for the Top 42 families, SVM gave the best performance, but it reached only 27.1 %. We can see that for the majority of cases SVM outperformed the other classification algorithms. We found that KNN, and GaussianNB constantly
performed worse compared to the other algorithms. After applying data augmentation (see section 3.5 for details), the performance has shown minor improvements, but still, all the models did not succeed in solving the classification problem (see chapter 4 for further details).

3.5 Data Augmentation

It is known that machine learning algorithms perform better with the availability of large datasets [42]. Given that our dataset is very limited in size, the dataset was augmented to produce a larger dataset. Data augmentation is a common practice in computer vision and machine learning which helps in increasing the size of the dataset and improving the performance of the models generated. Specifically, data augmentation helps in making the models less prone to the over-fitting [43, 44] which increases the generality of the model. Several data augmentation operations were performed over the dataset including flopping, cropping, and changing the brightness of the image. The different classes were augmented by a random selection of the augmentation methods, and they were augmented in a manner that balanced the number of data-points in each class which helped in solving the imbalance problem of the dataset. Figure 3.3 shows some examples of the augmentation performed on a sample image. Although data augmentation improved the models' accuracy such as in the case of Top 10 families where the best performance scored 35.7 % with data augmentation.
compared to 32.5% without data augmentation, these improvements were minor, and none of the models succeed in reaching a satisfying accuracy of the classifiers (see chapter 4 for the complete results). The same case follows for the other evaluation metrics. There was a slight improvement but it was not satisfactory.

3.6 Deep Learning

Testing with standard classifiers proved that such methods are not always successful in solving image classification and object recognition problems. This motivated us to use the state-of-art methods in images classification, namely deep learning.

In the simplest form of explanation, deep learning is a family of algorithms of multi-layer neural networks. Deep learning has been gaining significant popularity due to its success in improving state-of-the-art responses to several challenging problems such as image classification, speech recognition and object detection [3, 45, 46, 47]. Its success is highly linked to the recent advances in the computational power of machines and availability of large datasets [3, 48, 49] such as ImageNet [50], MNIST [51] and SVHN [52].

Deep learning methods are characterized as representation learning algorithms [3], meaning they have the ability to learn and extract the data features without explicit feature engineering from the user. They are typically built from multi-layer neural networks where multiple hidden layers of non-linear maps are gradually learning the feature representation of the data. Recurrent neural networks (RNNs) [53] and convolutional neural networks (ConvNets) [54] are among the most commonly used deep models in practice [3, 48, 49]. RNNs are commonly used in solving problems related to speech recognition and handling text [47, 55, 56], while ConvNets are known for their impressive performance in problems that deal with data that is images or videos, [6, 44, 57, 58], hence ConvNet was chosen as the primary building component for our tool. Below we give a brief explanation of what a ConvNet is, and what are the
Convolution neural networks (commonly known as ConvNets or CNNs) are believed to be one of the leading factors in improving and shifting the field of images, videos, and audio processing in recent years [48]. Although they were first introduced in 1989 [54], they did not gain their popularity until AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012 [44]. Since then there has been a tremendous amount of work to improve their performance and training procedure [46, 57, 59, 60].

ConvNets are multi-layer, feed-forward artificial neural networks that employ a mathematical operation called convolution in at least one of its layers [48]. One of their main characteristics is that they were designed to process data that is shaped in an n-dimensional array (e.g., images, signal languages and videos) [48, 3] which make them perform better compared to the models that were not designed with such consideration.

The main advantages of ConvNets over other models come from four properties, sparse interactions, parameter sharing, equivalent representations and the fact that they are multi-layer [48, 3] which allows ConvNets to scale to a large input size compared to the classical artificial neural networks (ANNs).

There are many components that can be found in a ConvNet. Below is a summary of the most common ones.

**Convolution layers** are the main building block in a ConvNet. They are built of learnable filters (i.e., kernels) [3] which the network aims to learn throughout the training. These filters have spatial dimensions (i.e., width, height) and a depth dimension, and different filters can be used in different parts of the network. During training, the filters are applied to the input in a similar manner of how a sliding window and a multiplication operation with the input value is performed with the
filters (see figure 3.4). Then the result of this multiplication is followed by a non-linear operation; we call an *activation function*, usually a Sigmoid or a rectifier. One noticeable difference between convolutions and classical MLP is that neurons in the convolutions are connected to a sub-region of the neurons in the previous layer which results in what is called local connection and that many of the neurons in a convolution layer share parameters \[3, 48\] rather than having different weights for each layer.

**Activation functions** are the non-linear functions that process the data after it has been multiplied with kernels. The most common activation functions are Rectified Linear Unit (ReLU) (see equation 3.8), Leaky Rectified Linear Unit (LeakyReLU) (see equation 3.9) and Tanh (see equation 3.10) which are the used functions in our ConvNets.

\[
f(x) = max(0, x) \tag{3.8}
\]

\[
f(x) = max(\alpha x, x) \tag{3.9}
\]

\[
tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.10}
\]

Fractional-strided convolutions layers (commonly known as deconvolutional layers \[61\]) are the main layers in building the generator network (discussed in 3.6.3). These layers run in reverse order than a typical convolutional layer. For
Figure 3.5: Example of max pooling and average pooling operations. In this example a 4x4 image is downsampling to a 2x2 by taking the maximum value or the average value of each sub-region.

example, if one built a neural network with fractional-strided convolutions layers as their primary building block, they can produce an image while if we use normal convolutional layers, we usually end up with classification scores.

**Pooling layers** are another important component of a ConvNet. In a nutshell, pooling is a function that downsamples the output of another layer. This helps in reducing the dimensionality of the spatial dimensions which reduces the processing time [3, 48, 49]. Fig 3.5 shows an example of the two most common pooling operations, the max pooling and the average pooling. In the max pooling, a rectangular part of the feature map is reduced to the maximum value inside it. The same operation is performed in average pooling, but the average is calculated instead of the maximum value.

**Batch Normalization** [62] is a recent method for adaptive reparameterization. This method helps in improving the overall performance of the network, having faster convergence and higher stability [48]. The main idea of batch normalization is that the input of each layer is the output of the previous layer and we can normalize that input to have a mean with a value that is approximately zero, and a standard deviation that is approximately one. Following the same notation in [48], let us assume that we
are using a ConvNet that is trained in batches of size $m$ then let the output of layer $i$ is $H$, then the input of layer $i + 1$ is defined as the following:

$$H' = \frac{H - \mu}{\sigma}$$  \hspace{1cm} (3.11)

where $\mu = \frac{1}{m} \sum H_i$ is the batch mean, and $\sigma = \sqrt{\delta + \frac{1}{m} \sum (H - \mu)^2}$ is the standard deviation of the batch, and $\delta$ is a constant added to the variance for numerical stability.

The importance of batch normalization comes from the fact that it helps to reduce the covariate shift. In simple terms, covariate shift is when a difference occurs between the training distribution and the testing distribution which results in making some errors by the model. Batch normalization works as if it was a regularizer of the network that reduces the effects of covariate shift. As discussed in section 3.6.3, batch normalization is essential in building the generator of a generative adversarial network.

**Dropout** \[[63]\], is a popular regularizing technique used in deep neural networks to prevent them from overfitting which was proven very effective \[[63, 48]\]. Assume that the dropout rate was chosen to be 0.5, Figure 3.6 shows an example of how dropout works during training. It is known that deep neural networks with a large number of parameters suffer from two main problems, overfitting and very long training time. Using dropout helps in reducing overfitting and training time. The main idea behind dropout is to randomly pick and deactivate neurons during training, which means we train on a partial number of the neurons where dropout is applied.

The last main component of a ConvNet is the **fully connected (FC)** layers. The fully connected layers (FC) are usually located at the top of the network, and they are the layers responsible for performing the classification task. There can be one or more fully connected layer, and the main feature of the fully connected layers
Figure 3.6: Dropout operation with drop rate 0.5

is that every neuron is connected to all the neuron (i.e., activations) in the previous
layer. In the last layer, the output from the previous layer is given as input to the
fully connected layer; then these layers flatten the given input to an N-dimensional
vector where N is the number of classes in the classification problem. The vector is
then passed to a classifier such as an SVM or a softmax layer that predicts the label.

The algorithm chosen for optimizing the network was the **Backpropagation**
Algorithm which is arguably the most common algorithm for training ConvNet.
Backpropagation has four stages that are repeated throughout the training, namely,
the forward pass, calculating the error, calculating the gradients and the backward
pass. In the forward pass, the input is passed throughout the network until the
calculation of the output vector in the classification layer. For example, starting from
the input layer $x$, it is multiplied by the weights and biases for each neuron in the
first hidden layer as:

$$h_{1ni} = w_{1i} * x + w_{2i} * x + ... + w_{ji} * x + b_{1i}$$ (3.12)

where $h_{1ni}$ is the output of neuron $i$ of layer 1, $w_{1i}$ is the first weights in neuron $i$, and
$b_{1i}$ is the bias of neuron $i$. In the second stage we calculate the error of the predicted
classes using a loss function. Softmax (shown in equation [3.13]) was chosen as the
classifier for all the networks used in this thesis along with cross-entropy as the loss
function (shown in equation [3.14]).
The third stage is calculating the gradient for the loss function, which is used to minimize the error. Based on the gradient calculated in the previous stage, the weights and the biases of the network are updated in the final stage, called backward pass.

3.6.1 Training a Convolution Neural Network from Scratch

A 5-layer ConvNet was trained from scratch for the classification problem (the network architecture is shown in figure 3.7). The weights and biases were randomly initialized then it was trained from 100 epochs for each different setting. We chose to fix the number of layers and test with different variations of the parameters, namely, we have tested with different learning rate, batch size and dropout rate. We found that a 0.0001 learning rate, a batch of size 16, and 0.5 as dropout rate gave the best results. Starting from this stage and for the following stages, the training data was with the augmented Fish dataset. It was clearly shown that this method is superior to the standard classification methods with an approximate improvement of 10% for the accuracy and great improvement for the other evaluation metrics.

However, with this shallow network and the random initialization, this ConvNet fails to solve the problem and did not reach an accuracy higher than 44% (as shown in the results chapter 4).

We then tested with using a deep, pre-trained network using the techniques of transfer learning (see section 3.6.2 for further details) which was proven better than training a network from scratch.
3.6.2 Transfer Learning

Due to the need of extensive training datasets for deep models and the high expense of training a model from scratch, it is common to use a pre-trained model over a large dataset and to modify the problem one needs to solve [64]. This technique is called, transfer learning, and it is one of the well-known practices in deep learning. Transfer learning has shown effectiveness in real-world applications such as image classification [65, 66], video summarizing and classification [57, 67], and text categorization and mining [68]. One can formally define transfer learning as the ability of a system to recognize knowledge and skills learned in previous tasks and apply them to novel tasks in new domains [69]. For example, if someone can drive a truck they can easily learn how to drive a car or if someone can speak a Latin-based language then they can learn English faster.

There are three typical settings of using transfer learning, namely inductive transfer learning, transductive transfer learning and unsupervised transfer learning [69]. We call the problem that the model was pre-trained on the source task, and the problem that we are trying to solve the target task. The different settings are based on the availability of labels in these sets and the similarity between the two problems. In inductive transfer learning, labeled data are available in a target domain, but they may or may not be available in the source domain. In transductive transfer learning, labeled data are available only in a source domain and they can be from different domains, but there is a single task or a single domain and a single task. Finally, on unsupervised transfer learning, there are no labeled data in both source and target domain.
Inductive transfer learning is the setting that was chosen for deployment in building the classifier. The assumption in the inductive transfer learning is that if \( n \) tasks are related to each other, then they may share some common features which can benefit all tasks.

One common application of transfer learning in deep learning is in image classification, where a deep ConvNet is pre-trained on a large dataset (e.g., ImageNet, which has 1.2 million images distributed over 1000 classes), and then the network can be used either as a fixed features extractor, or its weights are used as an initialization of the training on the actual or targeted task.

In the first case, the last layer (typically a fully connected layer) is removed from the network, and then the extracted features are fed to a non-linear classifier such as a soft-max classifier. Figure 3.8 shows an example of this case. The second way is to use the pre-trained network’s weights for the weight initialization, then continue with the training via the backpropagation algorithm.

Both methods were tested with the Fish dataset. Although fine-tuning is not recommended for small dataset [64, 70], we experimented with this method and as expected the model has fallen into over-fitting and gave worse performance than the one without fine-tuning.

Several models were tested as initial models for classification, including Inception v3 [71], Inception v4 [72] and VGG19 [58], all of which were pre-trained on ImageNet. We saw that VGG19 performed better than the other models, despite being the shallowest. However, using these models did not achieve high accuracy, but they had a better performance than using and training a model from scratch in the case of Top 10 families only and performed worse with the other categories in terms of accuracy. However, we also saw that using transfer learning improved the performance for the micro-recall and micro-precision. See chapter 4 for further details.
We conclude that the data we have is not enough for building a model using a supervised learning approach, even with using data augmentation and transfer learning. We explore using semi-supervised learning with synthetic unlabeled data which is discussed next.

### 3.6.3 Generative Adversarial Networks (GANs)

We believe that the limitation of our dataset caused the failure of using it in a supervised manner; hence, we moved to using a semi-supervised learning approach. The availability of more data even without labeling was a challenge in our case, and by the present data, we cannot solve the problem. Thus, we experimented with generating synthetic data to expand our dataset.

Generative models (see section 3.3 for a detailed explanation of generative models) are a family of machine learning algorithms that can be used in generating more data.

Amongst the different generative models, GAN stands as the model which is known for its excellent production of sharp, realistic images [43].
Generative adversarial networks (GANs) are unsupervised learning and generative models that consist of two competing networks: a generator component \( G(z; \theta^{(G)}) \) we call a *generator* or \( G \) and a discriminator \( D(G) \) component we call a *discriminator* or \( D \). \( D \) can be chosen as any differentiable function but typically is chosen as a ConvNet. Assuming that the two components are neural networks, GAN follows the following framework for its training: a random variable \( z \) (also called latent variable) is sampled from a normal or a uniform distribution and is input to \( G \). Then \( G \) aims to capture the data disruption and models \( z \) as one of the original dataset’s data point to the level where it fools \( D \) into thinking the generated data is from the original dataset. \( D \) is a neural network and a discriminative model that aims to build a strong classifier to distinguish between the real images and the fake ones (generated by \( G \)). The input to \( D \) are batches of images from the generated images from \( G \) and the real dataset. Figure 3.9, left, shows the general framework of a GAN. Then the output or the gradients of \( D \) is used to update the weights of \( D \) and \( G \).

The two components are trained via an adversarial process similar to a min-max game. Following the original approach in [73], Eq 3.15 shows the loss function for such framework.

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (3.15)
\]

GAN suffers from instability, and there is a possibility that the generator might fall in the case where it generates the same example in each iteration but since GANs were proposed in [73], GAN has gained high attention from the community, and there has been a significant effort of work on stabilizing and improving it as in [74, 75, 76, 77, 78, 79, 80, 81]. GANs have proven useful in several image processing and computer vision tasks, including image inpainting [82], style transfer [83], super-resolution [84, 85],...
Figure 3.9: Vanilla GAN is shown on the left and the updated GAN is shown on the right: in both cases the input to the discriminator and the generator are the same. The main difference is in the output on the discriminator, in vanilla GAN the discriminator outputs only whether an input in real (from the original dataset) or fake (generated from G).

hand pose estimation [86] and face recognition [87]. These success cases indicate that GANs have the potential to learn (in an unsupervised way) important underlying natural image properties that are exploited in turn to synthesize images.

Interestingly, generative models in general, and GANs, in particular, were proven to run well in a semi-supervised learning approach [88, 79, 78].

We adopt a deep convolutional generative adversarial networks (DCGAN) [89], an improved version of GAN, and updated it as proposed in [78] to update our original problem of the Fish classification.

Deep convolutional generative adversarial networks (DCGANs) is an improved version of GAN which usually produces good quality images with higher stability than GANs. As in a conventional GAN, DCGAN has two neural networks; a generator network (G) and a discriminator (D) network.

The generator network $G$ is a ConvNet but does not follow the standard architecture of ConvNets. The main difference is that it uses the fractional-strided convolutions layers (commonly known as deconvolutional layers) instead of pooling layers. The use of the fractional-strided convolutions layers makes $G$ works in a reverse order than the typical ConvNet. From the latent variable, it produces a data sample while
a regular ConvNet produces a class label from the input sample. The latent variable is randomly sampled from a uniformed or a normal distribution. The latent variable is passed through $G$’s layers which produce the desired output.

The results from $G$ are the input to $D$ alongside with the original data samples. The difference between the original setting of GAN and using it a semi-supervised learning is that $D_s$ output is modified to be real class 1, real class 2, ..., real class $k$, fake where $K$ is the number of classes from the original dataset. The semi-supervision part comes from the fact that GAN generates synthetic data that does not belong to a particular class, so $D$ is now trained in two settings, the supervised setting where is it trained to output to each class the input belongs to if the input is a real image and there the unsupervised part where it distinguished between a real image and a fake image. The advantage of using GAN is that it helps $D$ to generalize better thus being less prone to overfitting.

To explain this mathematically, assume the original dataset is of size $T$ and there are $K$ class labels in it. Let the generator network generate a set $S$ of $n$ images, that has an equal size to the original dataset and a label named $y = K + 1$. In supervised learning, the model outputs a K-dimension vector of logits that are then transformed into probability using a softmax function. The model is then trained by minimizing the cross-entropy between the true class label and the predicted class labels $p_{model}(y|x)$.

Let the model be updated to output K+1-dimension vector while using the new generated data. Furthermore, the probability corresponding to $x$ is fake is updated from $1 - D(x)$ as in the original framework to $p_{model}(y = K + 1|x)$. This implies that the unlabeled data can be used for learning with the condition that it belongs to one of $K$ classes. This is achieved by maximizing $log p_{model}(y \in \{1, ..., K\})$. For this case, the original loss function of GAN as in equation 3.15 is updated to equation 3.16. The notation here are followed from [78].
Let the new dataset have size $2S$ where half of our data set consists of real data and half of it is generated; then the loss function for training the classifier becomes:

$$L = -\mathbb{E}_{x,y \sim p_{\text{data}}(x,y)}[\log p_{\text{model}}(y|x)] - \mathbb{E}_{x \sim \mathcal{G}}[\log p_{\text{model}}(y = K + 1|x)]$$

$$= L_{\text{supervised}} + L_{\text{unsupervised}}$$

where

$$L_{\text{supervised}} = \mathbb{E}_{x,y \sim p_{\text{data}}(x,y)}[\log p_{\text{model}}(y|x, y < K + 1)]$$

and

$$L_{\text{unsupervised}} = -\mathbb{E}_{x \sim p_{\text{data}}(x)}[1 - p_{\text{model}}(y = K + 1|x)] + \mathbb{E}_{x \sim \mathcal{G}}[\log p_{\text{model}}(y = K + 1|x)]$$

(3.16)

As for GAN, the discriminator has the following architecture. It consists of 5 convolution layers where batch normalization and dropout were applied after every layer with the exception of the last layer where only batch normalization was applied. For the activation leaky ReLU was used. For the generator, we have a 3-layer generator of deconvolution layers with batch normalization and ReLU as the activation function for all layers except the last layer where Tanh was used. Batch normalization is a critical part of the GAN architecture because it prevents the generator from collapsing into generating a single example all the time.

The size $z$ was chosen to be 100 and it was sampled from a normal distribution. In order to improve the stability of GANs, feature matching was added to the implementation. Following the method in [78], the objective of the generator is modified to generate data that matches some real data statistics, meaning the generator is trained to output images with the same expected value to some intermediate layer from the discriminator. Formally the new objective of $G$ is

$$\|\mathbb{E}_{x \sim p_{\text{data}}(f(x))} - \mathbb{E}_{x \sim p_z(z)}(f(G(z)))\|_2^2$$

(3.17)
Figure 3.10: The discriminator architecture used in the GAN model. The network has 5 convolution layers and 19 layers in total. Batch normalization was used after every convolution layer except for the last layer and for the activation function leakyReLU was used in all the layers.

Figure 3.10 shows the discriminator architecture and Figure 3.11 shows the generator architecture.

This method achieves better accuracy in some cases, but the accuracy dropped significantly as the number of families increased. In the Top 10 case, the model scored an accuracy of 40 %, and for the Top 20 families, the accuracy dropped to 29.5 %. It achieved an accuracy of 25.4 % and 20.0 % for the Top 31 and Top 42, respectively. The network performance was better than traditional algorithms in the case of Top 10 families, but it did not perform well with other cases. Transfer learning in particular shows a superior performance compared to using semi-supervised learning with GAN. For example, in the 20 families case transfer learning achieves an accuracy of 36.7 % compared to 29.5 % in SSGAN.

In the next section, we discuss building the tool by combining transfer learning with using a GAN in a semi-supervised learning setting.
Figure 3.11: The generator architecture used in the GAN model. In total 11 layers were used. ReLU was used as the activation function for all the layer except the last layer where \textit{Tanh} was used.

### 3.6.4 Transfer Learning with Generative adversarial networks (GANs)

Using GAN in a semi-supervised learning approach proved to improve the accuracy of the model in some cases. However, that did not reach a significant level of accuracy. From earlier, it was shown that transfer learning improved the model accuracy, but the level of accuracy was not satisfactory. In the final stage of creating the tool, we look into improving the technique of transfer learning to achieve better results. Namely, the tool was designed by combining the approaches discussed in 3.6.3 and 3.6.2.

Similar to the approach of semi-supervised learning proposed in [78], we trained two ConvNets in an adversarial way. \( G \) has the same architecture as the one used in section 3.6.3 shown in figure 3.11. For \( D \), a modified version VGG19 [90] was used with the following differences. Batch normalization and dropout were applied after every convolution layer. LeakyReLU was chosen as the activation function for the convolution layer. The fully connected layers on the top of the network were removed and replaced with a global average pooling layer. This replacement was done following one of the best practices in image classification [91] which usually gives better results than using the normal fully connected layers and the last layer was a softmax function. The weights of \( D \) were initialized from a pre-trained VGG19 on ILSVRC-2012-CLS image classification.
Figure 3.12: Modified version of VGG19 used as the discriminator architecture with 70 layers in total. The layer box stands for a combination of four layers, a convolution layer, a leaky ReLU layer, a batch normalization layer, and a dropout layer.

Figure 3.13: Sample of GAN Synthetic Data dataset [92] which were retrieved from the tensorflow repository. $D$ architecture is shown in Figure 3.12.

Different parameter settings were tested for optimizing the model following the training approach explained in 3.2. We find that the model converges after 100 epochs of training. We also found that 0.0002 as learning rate gave the best result with a batch size of 16. Although it was suggested in the literature that training the generator in higher rate over $D$ gives a better result, we tested with that and found that it affects the performance negatively thus they trained both networks on the same rate. The size of $z$ was 100 as in the previous section.
Figure 3.14: The tool framework: the input are augmented then supplied to the discriminator network. A latent variable $Z$ is sampled and given to $G$ as an input. The output of $G$ is also supplied to $D$ as an input. The discriminator network is trained via propagation then the gradients are used to update the weights of the discriminator and the generator. After the training is finished, the discriminator is extracted and then used as the classification tool.

Combing a pre-trained model, fine-tuning it, and using GAN synthetic data for regularization has significantly improved the accuracy. We have reportedly reach an accuracy of 72% on the testing set. The tool also achieved significant results for the recall, precision, micro-recall and micro-precision compared to all the previous methods. Our results are shown in chapter 4 and Figure 3.13 shows a sample of the generated data. The final framework of the tool is shown in Figure 3.14.
Figure 3.15: Images representation throughout the network. (a) shows a raw image which an input to the network. (b) shows an early representation the network learns in the first layers. (c) shows the representation the network has in the top layers.

We were also interested to find what are the features the network extracts from the fish images. Figure 3.15.a shows a raw image given to the network. Then in figure 3.15.b, we can see an early representation of the image from one of the first layers. Finally, figure 3.15.c shows the features and the representation the network learned and generated. By inspecting the learned features, we believe that there many features that help to get high performance. The gut-shape, the eye shape, and the ratio between the height and width of the fish seem to be of the main features the classifier uses for the image classification.
Chapter 4

Experimental Evaluation

We explain here the experimental setup and give full details about the results of the experiments. The results presented here are the average of multiple runs of the models after parameters optimization.

4.1 Experiment setup

We ran our experiments on a local Linux machine running Ubuntu 14.04 and equipped with 28 Intel Xeon(R) CPU E5-2680 v4 with 125.8 GiB RAM. We also ran our experiments on Dragon Cluster on a mix of machines equipped with NVIDIA Tesla P100 and NVIDIA Quadro P6000. All the coding was done using Python and Tensorflow [93]. Tensorflow is an open-source interface that is optimized for deep learning algorithms.

4.2 Results

We list here the results of all the tested methods and algorithms in this thesis. The results are divided based on the algorithms used and the category of the dataset.

4.2.1 Standard classifiers

As discussed in chapter 3, we have experimented with several standard machine learning classifiers which are K-nearest neighbors (KNN), support vector machine (SVM), decision tree (DT), random forest (RF), multi-layer perceptron (MLP), Ad-
aBoost with the algorithm (algorithm=SAMME.R) and Gaussian Naïve Bayes (GaussianNB).

As shown below, the classifiers were trained on two versions of the Fish dataset; the first one is with augmentation and the second one is without augmentation.

Table 4.1: Standard Classifiers Results for the Top 10 Families

<table>
<thead>
<tr>
<th>Classifier</th>
<th>With augmentation</th>
<th>Without augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Precision</td>
</tr>
<tr>
<td>KNN</td>
<td>27.4 %</td>
<td>0.23</td>
</tr>
<tr>
<td>SVM</td>
<td>35.7 %</td>
<td>0.23</td>
</tr>
<tr>
<td>DT</td>
<td>35.7 %</td>
<td>0.18</td>
</tr>
<tr>
<td>RF</td>
<td>35.5 %</td>
<td>0.13</td>
</tr>
<tr>
<td>MLP</td>
<td>35.7 %</td>
<td>0.13</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>35.4 %</td>
<td>0.21</td>
</tr>
<tr>
<td>GaussianNB</td>
<td>3.3 %</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 4.2: Standard Classifiers Results for the Top 20 Families

<table>
<thead>
<tr>
<th>Classifier</th>
<th>With augmentation</th>
<th>Without augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Precision</td>
</tr>
<tr>
<td>KNN</td>
<td>21.7 %</td>
<td>0.17</td>
</tr>
<tr>
<td>SVM</td>
<td>28.2 %</td>
<td>0.19</td>
</tr>
<tr>
<td>DT</td>
<td>30.5 %</td>
<td>0.20</td>
</tr>
<tr>
<td>RF</td>
<td>30.4 %</td>
<td>0.14</td>
</tr>
<tr>
<td>MLP</td>
<td>30.5 %</td>
<td>0.09</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>29.9 %</td>
<td>0.14</td>
</tr>
<tr>
<td>GaussianNB</td>
<td>1.9 %</td>
<td>0.20</td>
</tr>
</tbody>
</table>
### Table 4.3: Standard Classifiers Results for the Top 31 Families

<table>
<thead>
<tr>
<th>Classifier</th>
<th>With augmentation</th>
<th>Without augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>18.1 % 0.15 0.17</td>
<td>26.7 % 0.24 0.26</td>
</tr>
<tr>
<td>SVM</td>
<td>28.2 % 0.15 0.28</td>
<td>27.7 % 0.11 0.25</td>
</tr>
<tr>
<td>DT</td>
<td>28.0 % 0.13 0.28</td>
<td>26.7 % 0.17 0.29</td>
</tr>
<tr>
<td>RF</td>
<td>28.1 % 0.08 0.28</td>
<td>26.1 % 0.13 0.26</td>
</tr>
<tr>
<td>MLP</td>
<td>30.5 % 0.09 0.30</td>
<td>27.1 % 0.14 0.27</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>27.5 % 0.11 0.28</td>
<td>25.4 % 0.11 0.16</td>
</tr>
<tr>
<td>GaussianNB</td>
<td>1.1 % 0.14 0.01</td>
<td>9.1 % 0.11 0.03</td>
</tr>
</tbody>
</table>

### Table 4.4: Standard Classifiers Results for the Top 42 Families

<table>
<thead>
<tr>
<th>Classifier</th>
<th>With augmentation</th>
<th>Without augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>19.6 % 0.27 0.27</td>
<td>16.6 % 0.14 0.16</td>
</tr>
<tr>
<td>SVM</td>
<td>27.5 % 0.13 0.22</td>
<td>27.1 % 0.13 0.27</td>
</tr>
<tr>
<td>DT</td>
<td>24.5 % 0.20 0.28</td>
<td>26.9 % 0.11 0.27</td>
</tr>
<tr>
<td>RF</td>
<td>30.6 % 0.16 0.29</td>
<td>27.0 % 0.14 0.27</td>
</tr>
<tr>
<td>MLP</td>
<td>28.7 % 0.14 0.27</td>
<td>27.0 % 0.07 0.27</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>26.3 % 0.04 0.12</td>
<td>25.8 % 0.17 0.27</td>
</tr>
<tr>
<td>GaussianNB</td>
<td>8.0 % 0.12 0.01</td>
<td>6.5 % 0.08 0.02</td>
</tr>
</tbody>
</table>

### 4.2.2 Training ConvNet from scratch

In this section we show the results of training a 5-layer ConvNet from scratch, meaning all the weights used in the network were randomly initialized. We train the network over 100 epochs. It was noticed that the network converges around the 70th epoch. As mentioned earlier in chapter 3, we used backpropagation as the training algorithm and we got the best results a mini-batch of size 16 and a learning rate of 0.0001.

### Table 4.5: ConvNet Results for the Top 10 Families

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43.7 %</td>
<td>0.34</td>
<td>0.41</td>
<td>0.50</td>
<td>0.51</td>
</tr>
</tbody>
</table>
Table 4.6: ConvNet Results for the Top 20 Families

<table>
<thead>
<tr>
<th>Top 20</th>
<th></th>
<th></th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38.4 %</td>
<td>0.33</td>
<td>0.35</td>
<td>0.37</td>
<td>0.37</td>
</tr>
</tbody>
</table>

Table 4.7: ConvNet Results for the Top 31 Families

<table>
<thead>
<tr>
<th>Top 31</th>
<th></th>
<th></th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32.4 %</td>
<td>0.28</td>
<td>0.24</td>
<td>0.29</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Table 4.8: ConvNet Results for the Top 42 Families

<table>
<thead>
<tr>
<th>Top 42</th>
<th></th>
<th></th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.6 %</td>
<td>0.19</td>
<td>0.21</td>
<td>0.26</td>
<td>0.29</td>
</tr>
</tbody>
</table>

4.2.3 Transfer learning

The results here show the performance of the use of the different pre-trained networks in a transfer learning setting. We experimented with 3 networks, Inception V3 [71], Inception V4 [72] and VGG19 [58]. Both Inception V3 and Inception V4 were designed by Google and their architectures are publicly released. VGG19 was designed by the Visual Geometry Group at University of Oxford and architecture is also publicly released. The weights of the network were retrieved from Tensorflow models’ zoo. The training was done in two settings; with fine-tuning and without fine-tuning (details about these settings can be found in chapter 3).
<table>
<thead>
<tr>
<th>Table 4.9: Transfer Learning Results for the Top 10 Families</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 10 Model</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Micro-precision</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4.10: Transfer Learning Results for the Top 20 Families</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 20 Model</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Micro-precision</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4.11: Transfer Learning Results for the Top 31 Families</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 31 Model</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Micro-precision</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4.12: Transfer Learning Results for the Top 42 Families</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 42 Model</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Inception V3</td>
</tr>
<tr>
<td>Inception V4</td>
</tr>
<tr>
<td>VGG19</td>
</tr>
<tr>
<td>Micro-precision</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Recall</td>
</tr>
<tr>
<td>Micro-recall</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Precision</td>
</tr>
<tr>
<td>Recall</td>
</tr>
</tbody>
</table>
4.2.4 Semi-supervised learning with generative adversarial network (SSGAN)

The results of using a GAN is semi-supervised learning manner are presented here. All the details regarding the architectures used can be found in chapter 3. The models were trained for 100 epochs and we found that a learning rate of 0.0002 gives the best results.

Table 4.13: SSGAN Results for the Top 10 Families

<table>
<thead>
<tr>
<th>Top 10</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>40.0 %</td>
<td>0.43</td>
<td>0.38</td>
<td>0.44</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 4.14: SSGAN Results for the Top 20 Families

<table>
<thead>
<tr>
<th>Top 20</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29.5 %</td>
<td>0.37</td>
<td>0.37</td>
<td>0.38</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table 4.15: SSGAN Results for the Top 31 Families

<table>
<thead>
<tr>
<th>Top 31</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25.4 %</td>
<td>0.30</td>
<td>0.32</td>
<td>0.34</td>
<td>0.37</td>
</tr>
</tbody>
</table>

Table 4.16: SSGAN Results for the Top 42 Families

<table>
<thead>
<tr>
<th>Top 42</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>Micro-precision</th>
<th>Micro-recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20.0 %</td>
<td>0.20</td>
<td>0.30</td>
<td>0.24</td>
<td>0.29</td>
</tr>
</tbody>
</table>
4.2.5 Semi-supervised learning with generative adversarial network (SSGAN) and transfer learning

This section shows the final results of the developed tool. We used a modified version of VGG19 combined with a GAN to achieve this level of accuracy. The weights of the network were retrieved from a pre-trained VGG19 on ImageNet. For each setting that we tested with, we trained the model for 100 epochs. We found that the network converges around the 60th epoch. We also found that using a learning rate of 0.0002 gives the best results.

Table 4.17: SSGAN with Transfer Learning Results for the Top 10 Families

<table>
<thead>
<tr>
<th></th>
<th>Top 10</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>Micro-precision</td>
<td>Micro-recall</td>
</tr>
<tr>
<td>72.1 %</td>
<td>0.849</td>
<td>0.781</td>
<td>0.89</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table 4.18: SSGAN with Transfer Learning Results for the Top 20 Families

<table>
<thead>
<tr>
<th></th>
<th>Top 20</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>Micro-precision</td>
<td>Micro-recall</td>
</tr>
<tr>
<td>63.4 %</td>
<td>0.702</td>
<td>0.693</td>
<td>0.77</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 4.19: SSGAN with Transfer Learning Results for the Top 31 Families

<table>
<thead>
<tr>
<th></th>
<th>Top 31</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>Micro-precision</td>
<td>Micro-recall</td>
</tr>
<tr>
<td>62.9 %</td>
<td>0.610</td>
<td>0.591</td>
<td>0.68</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Table 4.20: SSGAN with Transfer Learning Results for the Top 42 Families

<table>
<thead>
<tr>
<th></th>
<th>Top 42</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Precision</td>
<td>Recall</td>
<td>Micro-precision</td>
<td>Micro-recall</td>
</tr>
<tr>
<td>60.2 %</td>
<td>0.578</td>
<td>0.546</td>
<td>0.61</td>
<td>0.62</td>
</tr>
</tbody>
</table>
Chapter 5

Concluding Remarks

In this thesis we show that general framework researchers or developers can follow in developing classification tools for limited dataset. We show that using the standard machine learning algorithms fail to solve such a problem. On the other hand, deep learning methods and techniques were proven superior to the standard algorithms. We also show that regularization is an important factor for the success of deep learning. Dropout, batch normalization and using synthetic data are some of the techniques that were used in the developed tool. We have created a user-friendly tool that can be used by any user to classify Red Sea ichthyoplankton that reached a reported accuracy of 72.1%.

5.1 Future Research Work

The work presented in this thesis can be extended in the following directions. We want to update the tool such that the users can add more families to the tool and not being restricted to the existing one. We also plan to test with new and different methods of GAN that can be used for the synthetic data generation such as conditional GAN [94].
REFERENCES


[42]


6 Papers Submitted and Under Preparation