

# **Fauna Image Classification using Convolutional Neural Network**

**A Project Report**

*Submitted by*

**Adwait Pravin Aralkar  
Kavish Atul Sanghvi  
Saurabh Pankaj Sanghvi**

*Under the Guidance of*

**Prof. Ishani Saha**

*in partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY  
COMPUTER ENGINEERING**

**At**



**Mukesh Patel School of Technology Management & Engineering**

**July 2019 - April 2020**

## DECLARATION

I, Adwait Pravin Aralkar, Kavish Atul Sanghvi, Saurabh Pankaj Sanghvi, Roll No. B084, B085, B086, Bachelor of Technology in Computer Engineering, VII and VIII semesters, understand that plagiarism is defined as anyone or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.
2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)
3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source: IEEE, The institute, Dec. 2004)
4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.
5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the student:



Name: ADWAIT ARALKAR KAVISH SANGHVI SAURABH SANGHVI

Roll No.: B084 B085 B086

Place: Mukesh Patel School of Technology Management & Engineering, Mumbai

Date: 28<sup>th</sup> March 2020

# **CERTIFICATE**

This is to certify that the project entitled “Fauna Image Classification using Convolutional Neural Network” is the bonafide work carried out by Adwait Aralkar, Kavish Sanghvi, Saurabh Sanghvi, of Bachelor of Technology in Computer Engineering, Mukesh Patel School of Technology Management & Engineering (NMIMS), Mumbai, during the VII and VIII semesters of the academic year 2019-2020, in partial fulfillment of the requirements for the award of the Degree of Bachelors of Technology as per the norms prescribed by NMIMS. The project work has been assessed and found to be satisfactory.

---

Prof. Ishani Saha  
Internal Mentor

---

Examiner 1

---

Examiner 2

---

Dean  
MPSTME, NMIMS University

# Table of Contents

CHAPTER NO.	TITLE	PAGE NO.
	List of Figures	i
	List of Tables	ii
	Abbreviations	iii
	Abstract	iv
1.	INTRODUCTION	1
	1.1. Project Overview	1
	1.2. Algorithmic Overview	3
	1.3. Algorithm Selection Criteria	7
	1.4. Software Specifications	8
2.	LITERATURE SURVEY	9
	2.1. Convolutional Neural Network	9
	2.2. Transfer Learning	12
	2.3. Support Vector Machine	15
	2.4. K-Nearest Neighbor	17
	2.5. Random Forest Algorithm	19
	2.6. VGG16 with transfer learning and data augmentation	21
	2.7. Denoising Convolutional Neural Network	24
	2.8. Neural Network	26
	2.9. Summary of Literature Survey	28
3.	ANALYSIS & DESIGN	29
4.	IMPLEMENTATION	30
	4.1. Importing Libraries	30
	4.2. Creation of features/weights with VGG16	31
	4.3. Loading, training, and validating data	32
	4.4. Training model	33
	4.5. Graphing training and validation accuracy and loss	35
	4.6. Model evaluation	36
	4.7. Generating confusion matrix	37
	4.8. Testing images on model	38
5.	RESULTS & DISCUSSION	39
6.	CONCLUSION & FUTURE WORK	40
	REFERENCES	41
	PUBLICATIONS	42
	ACKNOWLEDGMENT	43

## List of Figures

CHAPTER NO.	TITLE	PAGE NO.
1.	INTRODUCTION	
	Fig 1.1. Block diagram of proposed fauna image classification using CNN	2
	Fig 1.2. 256x256 image before CNN can be applied	3
	Fig 1.3. CNN Classification Process	4
	Fig 1.4. SVM Classification	5
	Fig 1.5. KNN Algorithm	6
2.	REVIEW OF LITERATURE SURVEY	
	Fig 2.1. Layers showing the connection between layers	10
	Fig 2.2. Layers showing sharing of weights	10
	Fig 2.3. Processing the input feature with 32 filters and max-pooling	11
	Fig 2.4. Existing CNN architectures. (a) AlexNet (b) VGG16 (c) VGG19	13
	Fig 2.5. Proposed system block diagram. (a) CNN transfer learning (fine tuning) (b) CNN testing phase	13
	Fig 2.6. Overall comparison between the three CNN architectures and hybrid approach (SVM) over GHIM10K and CalTech256 database	14
	Fig 2.7. Illustration of optimal hyperplane	15
	Fig 2.8. Results of SVM and Neural Network experiments	16
	Fig 2.9. Similar data points exist close to each other	17
	Fig 2.10. (a) Neighborhoods (b) Shared Nearest Neighbors	18
	Fig 2.11. Decision Tree	19
	Fig 2.12. Random Forest Algorithm Layout	20
	Fig 2.13. Convolutional Neural Network Architecture	22
	Fig 2.14. VGG16 Architecture	23
	Fig 2.15. Image denoising in infocommunication system	24
	Fig 2.16. Algorithm for DnCNN model training	25
3.	ANALYSIS & DESIGN	
	Fig 3.1. Design and analysis plan	29
4.	IMPLEMENTATION	
	Fig 4.1. Importing necessary libraries	30
	Fig 4.2. Creating weights/features with VGG16	31
	Fig 4.3. Loading, training, and validating data	32
	Fig 4.4. Training model and model summary	33
	Fig 4.5. Graphing training, validation accuracy and loss	35
	Fig 4.6. Classification Matrices	36
	Fig 4.7. Generating Confusion Matrix	37
	Fig 4.8. Sample image output with accuracy and animal class	38

## **List of Tables**

<b>CHAPTER NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
2.	REVIEW OF LITERATURE SURVEY	
	Table 2.1. Results of activation function accuracy	11
	Table 2.2. Comparison between CNN architectures using average recall, precision and F-score on GHIM10k database	14
	Table 2.3. Comparison between CNN architectures using average recall, precision and F-score on CalTech256 database	14
	Table 2.4. Recognition rates for various numbers of hidden units in the hidden layer	27
	Table 2.5. Summary of Literature Survey	28
5	RESULTS & DISCUSSION	
	Table 5.1. Animal classes in the dataset	39

## Abbreviations

Abbreviation	Description
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
VGG	Visual Geometry Group
SVM	Support Vector Machine
KNN	K-Nearest Neighbor
RFA	Random Forest Algorithm

## **ABSTRACT**

Today, with the increasing volatility, necessity and applications of Artificial Intelligence, fields like Machine Learning, and its subsets, Deep Learning and Neural Networks have gained immense momentum. Convolutional Neural Network remains to be the most sought-after choice for computer scientists for image recognition & processing and is a type of ANN (Artificial Neural Network). The training needs softwares and tools like classifiers, which feed huge amounts of data, analyse them and extract useful features. These features are then used to observe a pattern and train the network to use similar data again the next time it is fed data. The applications include military applications, industrial applications and IoT applications which is why, we need classifiers to achieve maximum possible accuracy. Convolutional Neural Networks are used for image classification and recognition because of its high accuracy. This report provides an understanding of various image classification methods, literature review of different techniques, problem statements and comparisons of technologies that form the underlying base of our project; and reviews the results for the same. It also talks about the problem statement of our project - fauna classification and the methodology and supporting software tools to be used for the same. For example, Transfer Learning approaches, VGG16, TensorFlow, etc. We intend to train a convolutional neural network for efficiently classifying animal images with accurate results.



# Chapter 1

## Introduction

### 1.1. Project Overview

*Aim of the project was to develop an animal image classifier in dense forest environments to achieve desired accuracy, and aid ecologists and researchers in neural network/Artificial Intelligence & zoological domains to further study and/or improve habitat, environmental and extinction patterns.*

Today, with the increasing volatility, necessity and applications of Artificial Intelligence, fields like Machine Learning, and its subsets, Deep Learning and Neural Networks have gained immense momentum. It has become a data centric model. Developers are “training” the network to be “intelligent” and “independent”. The training needs softwares and tools like classifiers, which feed huge amounts of data, analyze them and extract useful features. These features are then used to observe a pattern and train the network to use similar data again the next time it is fed data. The applications include military applications, industrial applications and IoT applications which is why, we need classifiers to achieve maximum possible accuracy.

Efficient and reliable monitoring of wild animals in their natural habitats is essential to inform conservation and management decisions regarding wildlife species, migration patterns, habitat protection, and is possible, rehabilitation and grouping species of same animals together.

Processing a large volume of images and videos captured from camera traps manually – is extremely expensive, time-consuming and also monotonous. This presents a major obstacle to scientists and ecologists to monitor wildlife in an open environment.

In particular, we intend to use animal image dataset, and train a convolutional neural network, capable of classifying the image to a particular class with accuracy. This, in turn, can therefore speed up research findings, can also lead to discovery of potential new habitats as well as new unseen and/or rare species of animals (or on the verge of extinction) – within the same class, construct more efficient monitoring systems and subsequent management decisions, having the potential to make significant impacts to the world of ecology and trap-camera images analysis.

Images captured in a field represent a challenging task while classifying since they appear in a different pose, background clutter, different illumination and climatic conditions, human photographic errors can cause significant amounts of distortion, different angles, and occlusions. All these challenges necessitate an efficient algorithm for classification with most optimum accuracy. One of the biggest issues is ‘class imbalance’. Since there are uneven number of pictures for each sample, the algorithm could train some categories better, as compared to the others.

Scope of the project is to train a neural network for animal image classifier based on convolutional neural networks with accurate results, which can be used on animal image dataset, which need more memory allocation, good resolution, consistency in attributes and good classification.

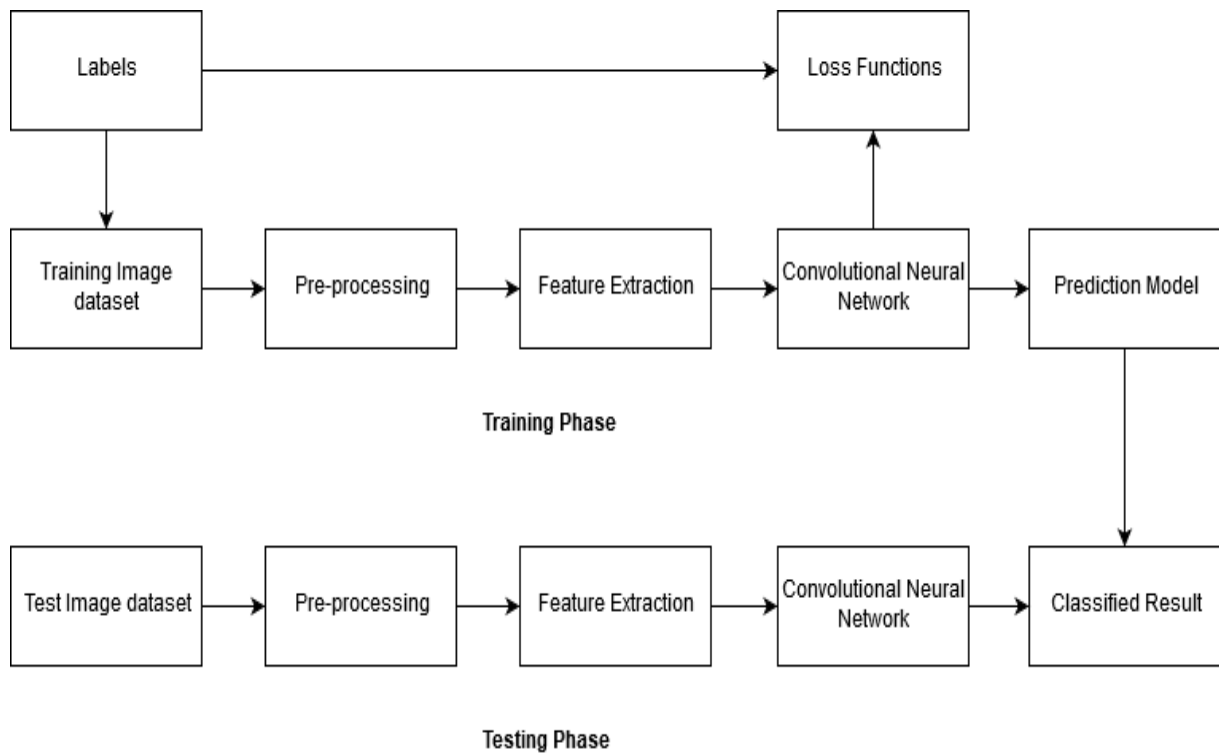


Figure 1.1. Block diagram of proposed fauna image classification using CNN

## 1.2. Algorithmic Overview

### 1.2.1. Convolutional Neural Network

The major idea behind Convolutional Neural Network is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a Convolutional Neural Network has just enough weights to look at a small patch of the image. It does not need to “look” at the entire image, a few major attributes are enough for it to classify the image. A real-life example can be looking at a page through a magnifying glass, readers can “see” the entire page, but focus only on the magnified, or the “important” part just like the Convolutional Neural Network.

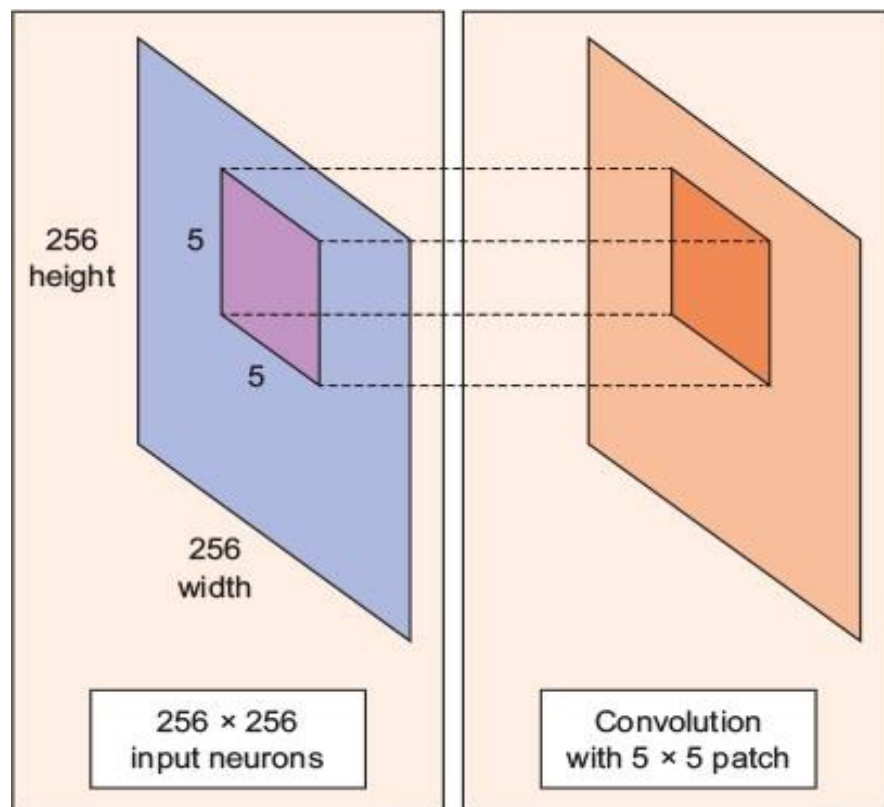


Figure 1.2. 256x256 image before CNN can be applied

In figure 1.2, Convolutional Neural Network[1] is applied on a 256 x 256 image. CNN efficiently scans it chunk by chunk. For example, consider a 5 × 5 window. The 5 × 5 window slides along the image (usually left to right, and top to bottom as convention). How fast it slides is called its “stride length”. For example, a stride length of 2 means the 5 × 5 sliding window moves by 2 pixels at a time until it spans the entire image. Weighted sum of the pixel values of an image is known as – convolution. As the window slides across the whole image, this convolution

process produces another image with a weight matrix. A typical CNN has multiple convolution layers. Convolution uses pooling for feature extraction. For example, if an image goes through a convolution layer on a weight matrix of  $5 \times 5 \times 64$ . It generates 64 convolutions by sliding a  $5 \times 5$  window. Therefore, this model has  $5 \times 5 \times 64 = 1,600$  parameters, which is remarkably fewer parameters than a fully connected network, which gives  $256 \times 256 = 65,536$ .

Convolutional Neural Network build their own features from raw signal, as opposed to other algorithms that use vector representations where every component usually makes some sense on its own. Pixels don't have meaning outside the context, but together they may contain more information about the object on a picture than a bunch of its properties that you feed into SVM.

The performance optimality of Convolutional Neural Network can be observed in the sense that the number of parameters is independent of the size of the original image. The number of parameters won't change in the convolution layer even if the same Convolutional Neural Network model is run on a  $300 \times 300$  image.

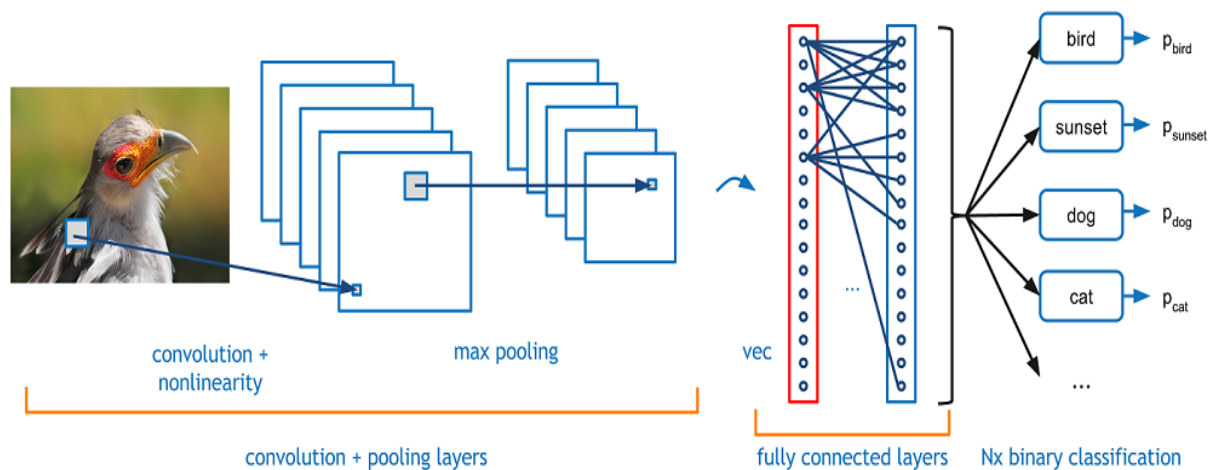


Figure 1.3. Convolutional Neural Network Classification Process

### 1.2.2. Support Vector Machine Classification Algorithm

Support Vector Machine (SVM)[4] is fundamentally a binary classification algorithm. It also falls under the umbrella of Machine Learning. Feature extraction is the most crucial, time consuming and complex step in SVM. The principle behind SVM is simple: The algorithm creates a line or a “hyperplane” which separates the data into classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible.

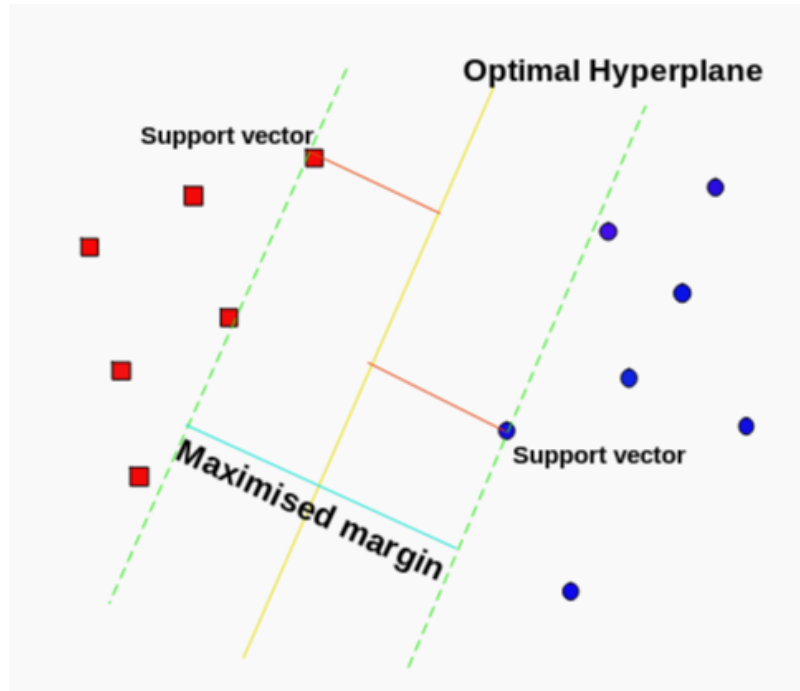


Figure 1.4. SVM Classification

According to the SVM algorithm we find the points closest to the line from both the classes. These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane. In figure 1.4., red and blue entities are 2 different datasets that need to be classified and are scattered throughout. and SVM find an optimal “hyperplane” (yellow line), which discriminates between both classes, thereby classifying them into different categories efficiently. It can classify non-linear data as well by using 3-D systems.

### 1.2.3. K-Nearest Neighbor Classification

K nearest neighbor[5] is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g.- distance functions). KNN algorithm fares across all parameters of considerations. It is commonly used for its ease of interpretation and low calculation time. Figure 1.5. illustrates the mechanism of KNN algorithm.

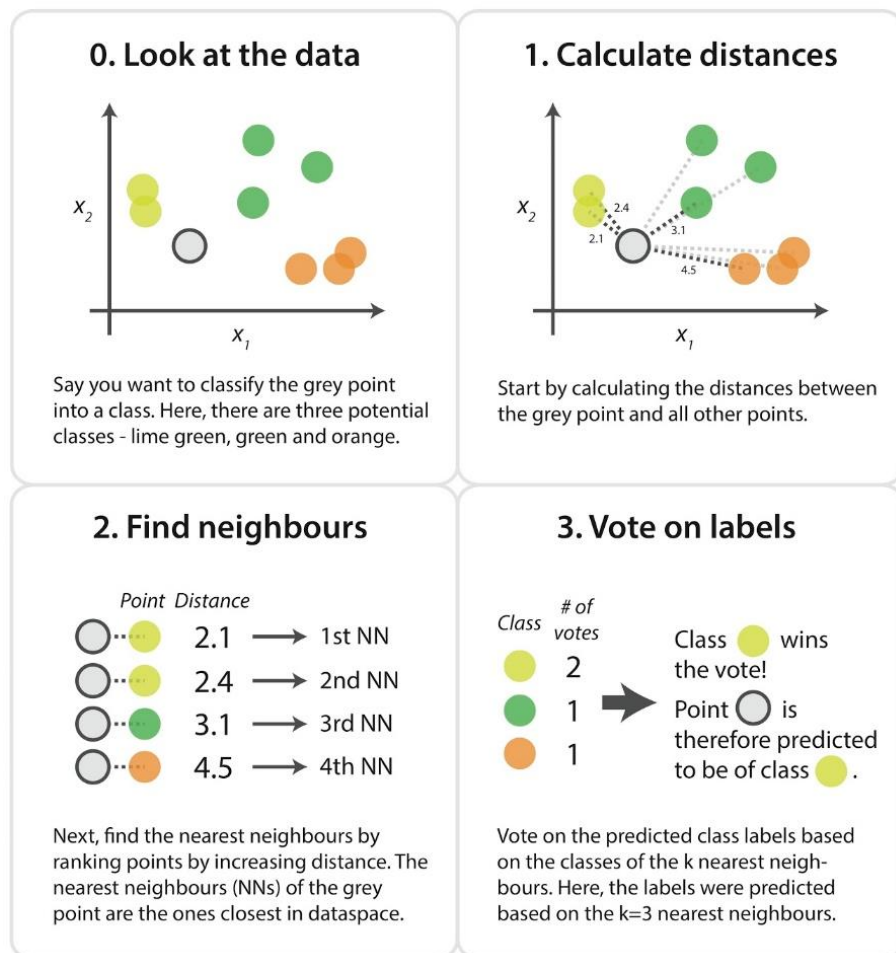


Figure 1.5. KNN Algorithm

### 1.3. Algorithm Selection Criteria

Convolutional Neural Network can be thought of as an automatic feature extractor for the image. Now, if we use algorithm with pixel vector (SVM), we lose a lot of spatial interaction between pixels, whereas a Convolutional Neural Network effectively uses adjacent pixel information to efficiently down sample the image first by convolution and then uses a prediction layer at the end. Feature selector algorithms for extraction are also not needed in Convolutional Neural Network. It covers both, local as well as global features well, whereas algorithm-based classification, sometimes only either global/local features work well. The work of a convolutional neural network is usually interpreted as a transition from specific image features to more abstract details, and further to even more abstract details, up to highlighting high-level concept [4]. Furthermore, instead of being an algorithm, it is a neural network, which can be easily and efficiently scaled to large datasets unlike other algorithms. KNN and Convolutional Neural Network perform competitively within their respective algorithms, but CNN produces higher accuracy than KNN even though KNN is easier to implement; and hence chosen as a better approach. Good generalization and invariance to local fluctuations makes Convolutional Neural Network highly scalable. Lastly, Convolutional Neural Network store much more information (as parameters) than SVM, K-NN or Random Forest methods. Hence, we chose to implement Convolutional Neural Network in our project.

## 1.4. Software Requirements and Specifications

Programming Language: Python

IDE: Jupyter Notebook

Libraries:

- Panda, for providing high-performance, easy-to-use data structures and data analysis
- NumPy, for mathematical and logical operations on arrays can be performed
- Keras API, to enable fast experimentation with deep neural networks

Dataset: Animal-10, which contains around 26179 hand-picked images of animals such as Butterfly, Cat, Chicken, Cow, Dog, Elephant, Horse, Sheep, Spyder, and Squirrel. Image count for each category varies from 2000 to 5000 images. Made available open-source through Kaggle[10]



## **Chapter 2**

### **Literature Survey**

#### **2.1. Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning**

The authors of this paper[2] recommend ways for automated classification of underwater fish species. The proposed method suggested implementation of removing the noise in the dataset. Application of Image Processing before the training step helps to remove the underwater obstacles, dirt and non-fish bodies from the images. The second step uses Deep Learning approach by implementation of Convolutional Neural Network for the classification of the Fish Species. It compares ReLU, SoftMax and tanh activation functions and found ReLU to be the most accurate.

A high accuracy fish classification is required for greater understanding of fish behavior in Ichthyology and by marine biologists. Majority of available methods focus on classification of fishes outside of water because underwater classification poses challenges such as background noises, distortion of images, the presence of other water bodies in images, image quality and occlusion. The method proposed uses a novel technique based on Convolutional Neural Networks, Deep Learning and Image Processing to achieve an accuracy of 96.29%.

The proposed method uses Convolutional Neural Networks which makes the process simpler and more robust even while working with a large dataset. Convolutional Neural Networks are also much more flexible and can adapt to the new incoming data as the dataset matures. The authors make use of the fish dataset from the Fish4Knowledge project for testing the algorithm. They perform the classification by pre-processing the images using Gaussian Blurring, Morphological Operations, Otsu's Thresholding and Pyramid Mean Shifting, further feeding the enhanced images to a Convolutional Neural Network for classification.

The authors of this paper proposed a methodology for the discrimination of fish species. The initial step taken by the system aims at removing the noise in the dataset. The second step uses Deep Learning approach by implementation of Convolutional Neural Networks for the classification of the Fish Species. With the implementation of Otsu's thresholding, the image

provides a sure foreground with the fish in focus. Next step deals with the implementation of Morphological Operations, viz-a-viz, Erosion and Dilation of the binarized image. The Second step of the procedure is the implementation of a Convolutional Neural Network for classification of Fish species. The input layer of the network takes the 100x100x3 original RGB image stacked with the 100x100x1 image which is the output of the pre-processing stage, thus making the input of 100x100x4, the fully-connected layer where we get the trained output and the intermediate hidden layers. The network has a series of convolutional and pooling layers. Neurons in layer say, 'm' are connected to a subset of neurons from the previous layer of (m-1), where the (m-1) layered neurons have contiguous receptive fields, as shown in Figure 2.1.

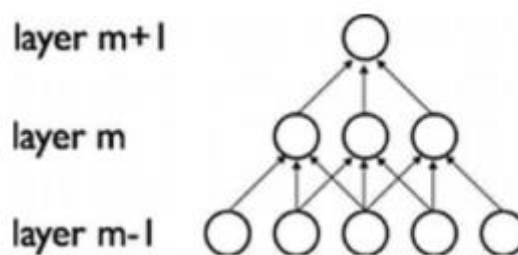


Figure 2.1. Graphical flow of layers showing the connection between layers.

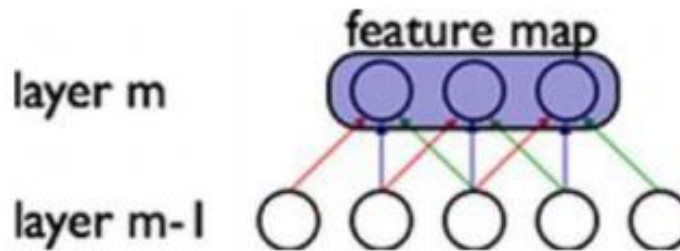


Figure 2.2. Graphical flow of layers showing sharing of Weights.

Figure 2.2. represents three hidden units. The weights of similar color are shared, thus are inferred to be identical. In addition to this, weight sharing tends to decrease the number of free learning parameters. Due to this control, Convolutional Neural Network tends to achieve better generalization on vision problems. The max-pooling layers act as non-linear down sampling, in which the input image is partitioned into non-overlapping rectangles. The output of each sub-region is the maximum value. The Convolution Layer is the first layer of the CNN network. The structure of this layer is shown in figure 2.3. It consists of a convolution mask, bias terms and a function expression. Together, these generate the output of the layer. Figure 2.3. shows a 5x5x4 mask that performs convolution over a 100x100x4 input feature map. Upon application of 32 such 5x5x4 filters, the resultant output is a 96x96x32 matrix.

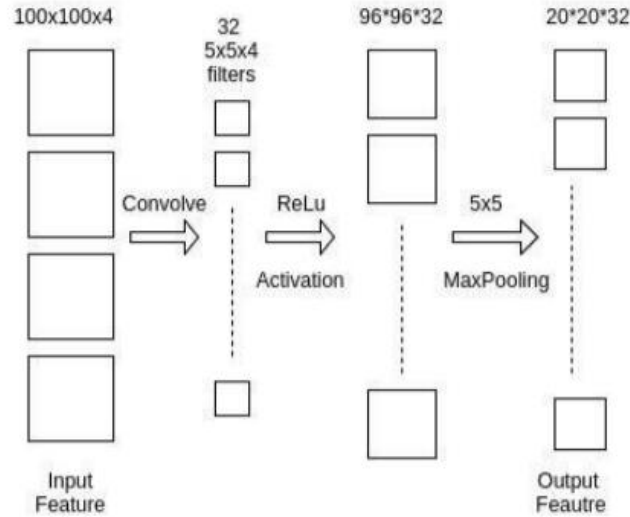


Figure 2.3. Processing the input feature with 32 filters and max-pooling.

The next layer in the network is a subsampling layer. The Subsampling layer is designed to have the same number of planes as the convolution layer. The purpose of this layer is to reduce the size of the feature map. It divides the image into blocks of 5x5 and performs max-pooling. Sub-sampling layer preserves the relative information between features and not the exact relation. Figure 2.3. shows how the input features are processed with 32 filters and max-pooling. The above process is repeated two times again once with 64 filters and then with 32 filters. The final output is connected to a fully connected layer which is further connected to an 80% dropout layer and lastly another fully connected layer which classifies the images into appropriate categories. The aim of the training algorithm is to train a network such that the error is minimized between the network output and the desired output. In the proposed method, we provide the comparison of different Activation Functions that will be applied to the different Layers in the CNN. The following are the three Activation functions, namely, a) ReLU b) Sigmoid c) tanh

The proposed method was tested in Python on the dataset Fish4Knowledge of 27,142 images Table 2.1. shows the results as the accuracy of the correctly predicted test images in the sample as given by equation shown below:

Activation Function used	Overall accuracy
ReLU	96.29%
tanh	72.62%
Softmax	61.91%

Table 2.1. Results of activation function accuracy

The authors plan to improvise the algorithm further by implementing Image Enhancement techniques to counter for the lost features in the images.

## 2.2. Transfer Learning for Image Classification

Convolutional Neural Network gained great attention for robust feature extraction and information mining. CNN had been used for variety of applications such as object recognition, image super-resolution, semantic segmentation, etc. By keeping constant the baseline learning topology, various CNN architectures were proposed to improve the respective system performance. Among these, AlexNet, VGG16 and VGG19 are the famous CNN architecture introduced for object recognition task. The authors of this paper[3] make use of transfer learning to fine-tune the pre-trained network (VGG19) parameters for image classification task. Further, performance of the VGG19 architecture is compared with AlexNet and VGG16. The authors have used two state-of-the-art databases namely: GHIM10K and CalTech256 to study the effect of CNN architecture for robust feature extraction, effect of deeper network by comparing the result with AlexNet and VGG16 architectures for image classification task. Performance evaluation has been carried out using average recall, precision and F-score. Performance analysis shows that fine-tuned VGG19 architecture outperforms the other CNN and hybrid learning approach for image classification task.

AlexNet was proposed to solve the object recognition problem. It was the first try to learn the network parameters for recognition task over very large-scale database. AlexNet consists of twenty-six layers, out of which last two layers are SoftMax and output layers. Network architecture is divided into three parts. Figure 2.4.(a) shows the network architecture. First part of the network consists of two units, each unit comprises {convolution, ReLU, normalization and pooling layer}. Second part of the network consist of four units, each of them comprises {convolution and pooling layer}. Last part of the network corresponds to the non-linear activation unit which corresponds to fully connected (FC) layers, ReLU and drop-out layer. Accuracy of the CNN architecture highly depends upon the three factors namely: Large scale database, high end computational unit and the network depth. Figure 2.4.(b) shows the network architecture. Unlike AlexNet, VGG16 consists of replicative structure of {convolution, ReLU and pooling layer}. They increased the number of such network unit to design deeper network. However, considered smaller size receptive window for each convolutional filter as compared to AlexNet. Non-linear activation unit is same as that of AlexNet. Further, deeper network VGG19 is proposed for the same task (Object detection). VGG19 comprises of some extra convolutional ReLU units in the middle of the network as compared to VGG16. However, this minute change in the architecture turns into the accuracy enhancement for object recognition task.

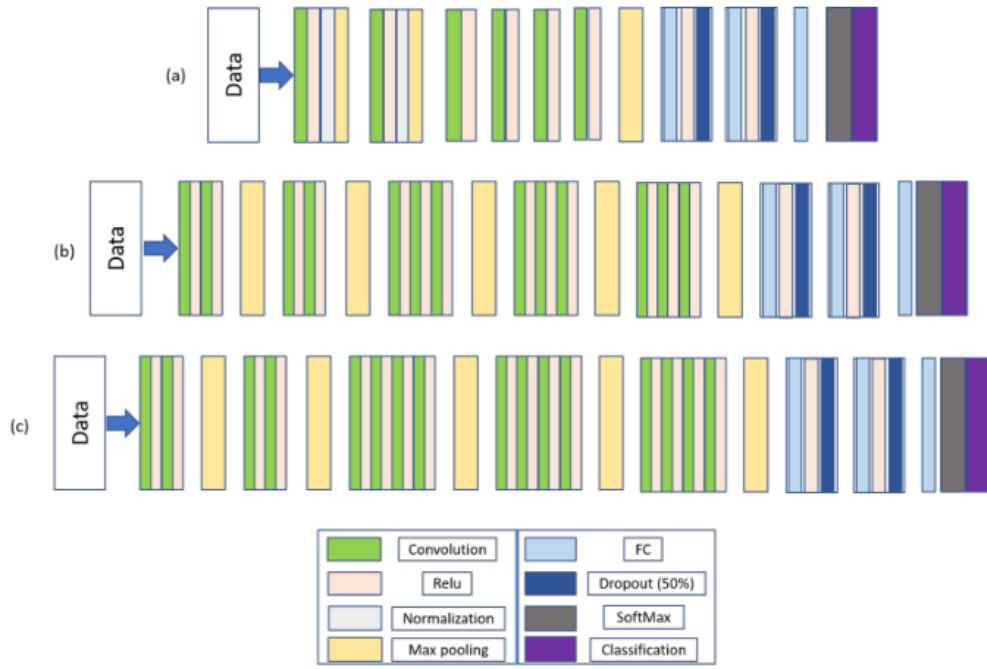


Figure 2.4. Existing CNN architectures. (a) AlexNet (b) VGG16 (c) VGG19

In this work, the authors have fine-tuned the network parameters of the VGG19 over two databases namely: CatITech256 and GHIM10K. Proposed network is divided into two parts, (1) CNN training phase and (2) CNN testing phase. Figure 2.5. shows the proposed system flow. Figure 2.5.(a) illustrates the CNN training phase in which network parameters of the VGG19 are fine-tuned and trained VGG19 is obtained. However, Figure 2.5.(b) shows the CNN testing phase which comprises the test image followed by trained VGG19 to estimate the image class probability.

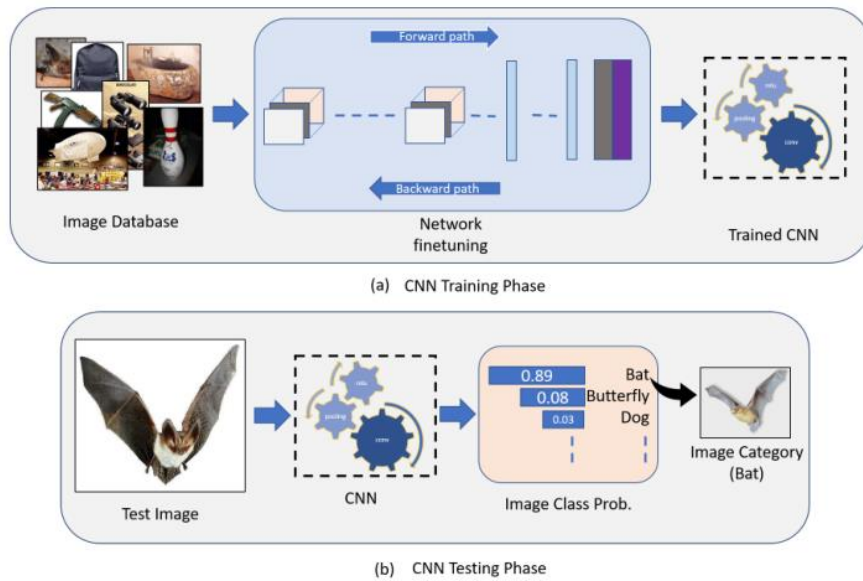


Figure 2.5. Proposed system block diagram. (a) CNN transfer learning (fine tuning) (b) CNN testing phase

Experimental results, the authors have performed two experiments on two state of the art databases namely: GHIM10K and CalTech256.

Experiment 1: The authors have carried this experiment on publicly available GHIM10K database. It consists of 20 classes; each class is having 500 images. In this experiment, the authors have analyzed the performance of VGG19 architecture for image classification task. Along with VGG19, the authors have analyzed performance of AlexNet and VGG16 on GBHIM10K database. Table 2.2 shows the comparison between CNN architectures using average recall, precision and F-score on GHIM10K database. Table 2.2 witnessed to the improvement in the accuracy due to the VGG19 architecture.

Method	Recall	Precision	F-Score
AlexNet	96.88	96.56	96.72
VGG16	98.57	98.23	98.40
VGG19	99.38	99.23	99.30

Table 2.2. Comparison between CNN architectures using average recall, precision and F-score on GHIM10k database

Experiment 2: This experiment comprises use of CalTech256 database for performance evaluation of VGG19 architecture for image classification task. CalTech256 consists of 256 categories, each class is having minimum 80 images. Due to the space limit, it is not possible to show the class wise accuracy, as there are 256 different categories. Table 2.3. shows the comparison between CNN architectures using average recall, precision and F-score on CalTech256 database. From Table 2.3, it can be observed that VGG19 improves the system accuracy. Figure 2.7. shows the overall comparison between the three CNN architectures over GHIM10K and CalTech256 datasets.

Method	Recall	Precision	F-Score
AlexNet	96.88	96.56	96.72
VGG16	98.57	98.23	98.40
VGG19	99.38	99.23	99.30

Table 2.3. Comparison between CNN architectures using average recall, precision and F-score on CalTech256 database

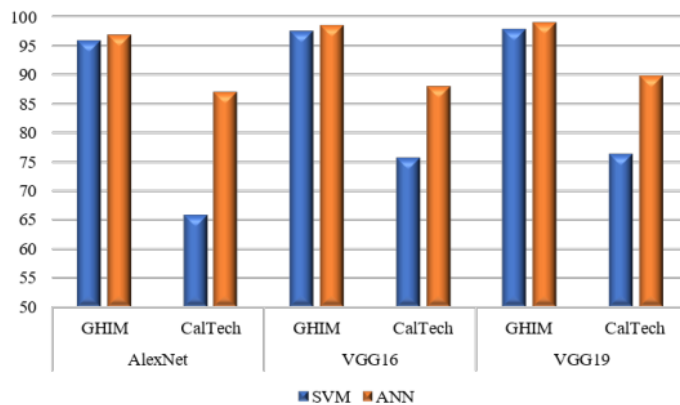


Figure 2.6. Overall comparison between the three CNN architectures and hybrid approach (SVM) over GHIM10K and CalTech256 database

### 2.3. Image Classification via Support Vector Machine

The paper[4] compares the performance of Support Vector Machine with Neural Network. The neural networks cannot escape from its own limitations including the local optimum or the dependence on the input sample data. The authors suggest to implement another algorithm named support vector machine, whose main idea is to build a hyperplane as the decision surface, is introduced to solve the problems. The algorithm creates a line or hyperplane which separates the data into classes. Support vector machine is a sophisticated and powerful algorithm. The results of the experiments proved that the accuracy of the classification by support vector machine is more excellent than the neural networks. The support vector machine reveals better effect on the single classification than neural networks.

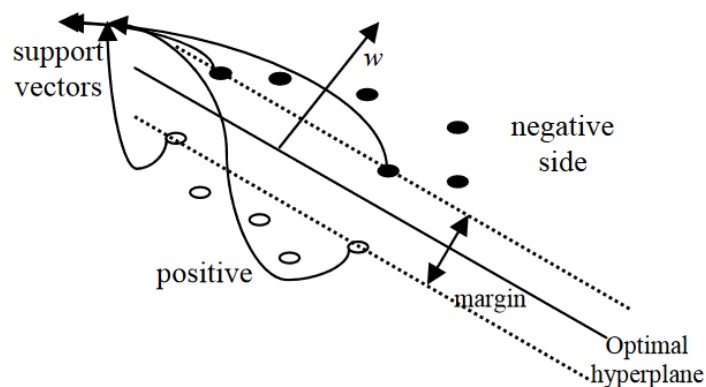


Figure 2.7. Illustration of optimal hyperplane

Image classification has been a main problem, and most of researchers are the concerned with the neural networks to realize the images classification. However, the neural networks cannot escape from its own limitations including the local optimum or the dependence on the input sample data. In this paper[4], the authors implement another algorithm named support vector machine, whose main idea is to build a hyperplane as the decision surface, is introduced to solve the problems. In order to solve the optimal hyperplane for the separable patterns problem, the method of Lagrange multiplier is transformed into its dual problem. It is proved that the support vector machine can solve the problem of classification perfectly, with regard to the input data, the eigen values of the images gray information are treated by the method of Principal Component Analysis and are abstracted as input sample. Support Vector Machine (SVM) is fundamentally a binary classification algorithm. It also falls under the umbrella of Machine Learning. Feature extraction is the most crucial, time consuming and complex step in SVM. The principle behind SVM is simple: The algorithm creates a line or a “hyperplane” which separates the data into classes. Support vector machine is a sophisticated and powerful algorithm.

SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible. According to the SVM algorithm we find the points closest to the line from both the classes. These points are called support vectors. Now, compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane. In figure 2.7., entities are 2 different datasets that need to be classified and are scattered throughout. and SVM find an optimal “hyperplane”, which discriminates between both classes, thereby classifying them into different categories efficiently. It can classify non-linear data as well by using 3-D systems.

Experiments results and analysis: Some figures were divided into 5 categories are taken into the experiment, where it proves that the support vector machine can solve the problem of classification more perfectly than the neural networks. Firstly, the gray value, which ranges from 0 to 255, of the image is constructed as the input vector  $\alpha$  i 48 images are shown in those experiments, so 48 different input vectors are provided, all of which are expressed as  $\alpha I = [\alpha 0, \alpha 2, \dots, \alpha 255]$

The results of the experiments are obvious that the accuracy of the classification by support vector machine is more excellent than the neural networks. On the whole, the support vector machine’s accuracy for these 5 patterns is 89.66%, which is far higher than the neural networks 41.38%. And the support vector machine reveals better effect on the single classification as shown in Figure 2.8.

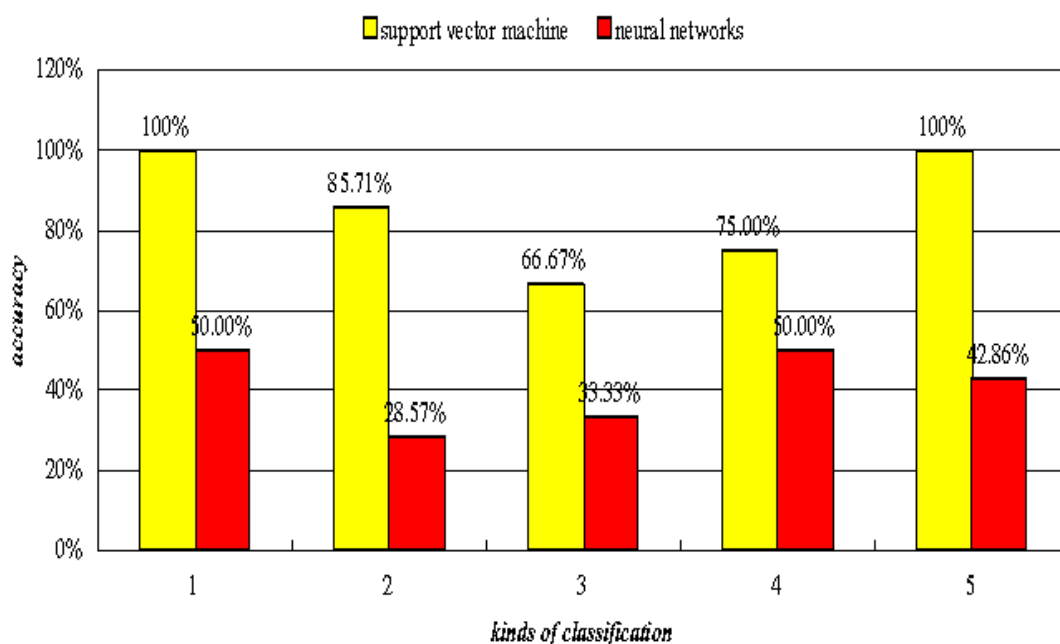


Figure 2.8 Results of SVM and Neural Network experiments



## 2.4. A KNN Research Paper Classification Based on Shared Nearest Neighbor

The paper[5] discusses about implementation of K-Nearest Neighbors for text categorization which based on shared nearest neighbor, combining the BM25 similarity calculation method and the Neighborhood Information of samples. The usefulness of this method has been fully verified in the NTCIR-8 Patent Classification evaluation. It obtained the best score in the English patent classification task, but the results are still not ideal as there are few considerations that can be corrected for better results. Firstly, corpus processing is too rough without careful feature selection, and the large feature space confused the topic information of patent, which consequently weakened performance of the system to a certain extent. Secondly, the problem of uneven density of corpus is not considered, resulting in wrong category decision-making for topics whose training data is inadequate. However, the system considered samples neighborhood information to amend the weight of each search result so as to objectively assign higher weight to the sample which is more similar with the topic, and at the same time avoided the phenomenon that the similar samples with lower ranks are severely punished because of the location. This strategy is more reasonable for category decision-making, and it is worth further investigating for patent classification.

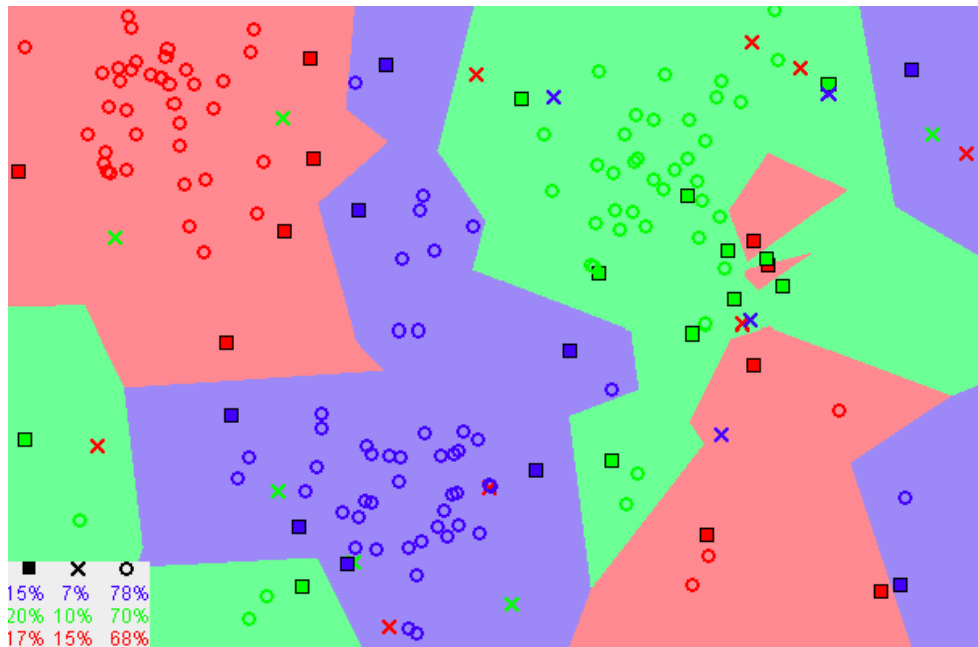


Figure 2.9. Similar data points typically exist close to each other

The paper compares different classification algorithms such as SVM which is a two-category classification model, Naive Bayes which is unable to classify with large class space and the hardware used. Similarity among samples is the key part of the KNN algorithm, BM25

similarity calculation method, is a bag-of-words retrieval function, combines the word frequency and document frequency, balances the length of the document, and is a highly efficient similarity calculation method. It also compares Shared Nearest Neighbour, here the  $link(p_i, p_j)$  is the number of shared neighbors between the sample  $p_i$  and  $p_j$ , which is a shared nearest neighbor concept.

Formula:  $link(p_i, p_j) = |\{x | S_i \cap S_j\}|$

The graphics below of different shapes represent samples of different categories, if the neighborhood radius is set to 4, the neighborhoods of  $p_1$ ,  $p_2$ ,  $p_3$ , and  $p_4$  are roped above, the number of shared neighbors between  $p_0$  and  $p_1$ ,  $p_2$ ,  $p_3$  are all 2,  $p_0$  and  $p_4$  shared 0 neighbors, obviously  $p_0$  and  $p_1$ ,  $p_2$ ,  $p_3$  has a certain similarity, the introduction of  $link$  to measure the similarity between samples can fully reflect the neighborhood information. Larger link means that samples contain more similar properties.

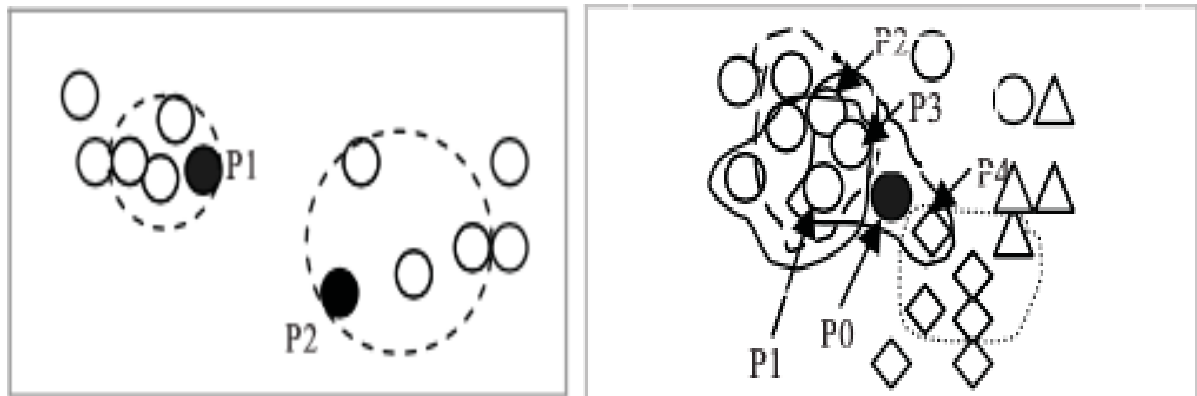
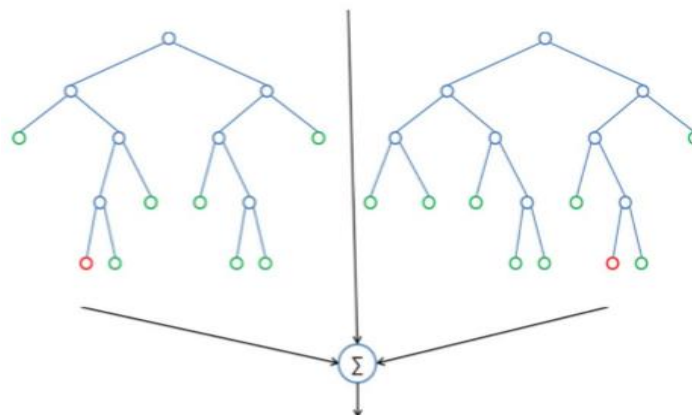


Figure 2.10. (a) Neighborhoods (b) Shared Nearest Neighbors

## 2.5. Credit Card Fraud Detection using Random Forest Algorithm

The paper[6] dealt with credit card fraud detection in real world. Online fraudulent transaction activities are increasing day by day. Hence, the need for more effective methods is growing at an exponential rate and existing algorithms are unable to cover all fraud activities. The fraudsters are looking for sensitive information such as credit card number, bank account and other user details in order to perform transactions and gain money or sensitive information. Thus, credit card fraud has become the major issue in today's technological world which has a massive problem in bank transactions. There are many fraud transactions which cannot be easily identified by the user and also by the banking authority which leads to loss of sensitive data. There are various models which are used for detecting the fraud transactions based on the behaviour of the transactions and these methods can be classified as two broad categories such as supervised learning and unsupervised learning algorithm. In existing system for finding the accuracy of the fraudulent activates they have used methods such as Cluster Analysis, Support Vector Machine, Naïve Bayer's Classification etc.

Research and literature review of existing system showed that in a case study involving credit card fraud detection, where data normalization was applied before Naïve Bayer's and Cluster Analysis and with results obtained from the use of these methods on fraud detection has shown that by clustering attributes neuronal inputs can be minimized and promising results can be obtained by using normalized data. This research was based on unsupervised learning. Significance of this paper was to find new methods for fraud detection and to increase the accuracy of results. The data set for this paper is based on real life transactional data by a large European company and personal details in data have been kept confidential. Accuracy of this previous algorithm is around 50%. Thus, the accuracy of the results obtained from these methods were less - when compared with the proposed system.



In proposed system, Random Forest Algorithm (RFA) was used for finding the fraudulent transactions and the accuracy of those transactions. It is based on supervised learning algorithm where it uses decision trees for classification of the dataset. Then the dataset was split into two categories as Trained dataset and Testing dataset for comparing and analysing the dataset. By applying the Random Forest Algorithm, the dataset was classified into four categories which were obtained in the form of a confusion matrix. Based on the above classification of data, performance analysis was done. In this analysis, the accuracy of credit card fraud transactions was obtained which is finally represented in the form of graphical representation. The confusion matrix basically takes and collates most accurate values directly from different decision trees, thereby reducing computational complexity of making multiple decision trees. This Random Forest Algorithm is based on supervised learning and the major advantage of this algorithm is that it can be used for both Classification and Regression.

During analysis process all the duplicate values and also the null values will be removed from the dataset. Now the dataset will be pre-processed based on the amount and transaction time for finding the accuracy of the resultant dataset. Here for dataset classification, the authors used a software called 'Scikit-learn'. After the pre-processing of the dataset, the Random Forest Algorithm was applied. By applying RFA, the pre-processed dataset was analysed again and then a confusion matrix was obtained. In confusion matrix, the dataset got partitioned into four blocks as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). Post this operation, the data-set was partitioned continuously until all the data was validated.



Figure 2.12. Random Forest Algorithm Layout

## **2.6. VGG16 for plant image classification with transfer learning and data augmentation**

Plants are important in earth's ecosystem and known as the backbone of all life on earth. They are crucial for overall balance of ecology in an area. They help to give off oxygen and use carbon dioxide while they are undergoing photosynthesis process which benefits tremendously for humans and our atmosphere. Good understanding of plants is crucial especially in identifying rare and new plant species. Plant identification is known to be a challenging and demanding task, mainly due to the large number of plant species. The traditional plant identification method normally done by expert botanists, which involves manual measurement for feature of plants is known to be a very tedious, slow and expensive task. Some features of plant that is known to be useful to identify the species is the leaves, fruits, seeds, flowers and bark. Since, the past decade, there has been an expansive development and advancement of technology that can be used to identify the plant as a faster and cheaper alternative. The advancement of deep learning, especially for computer vision benefits tremendously in identification and classification of plant species.

VGGnet model[7] used in this paper is a pretrained model, which has been trained previously in ImageNet dataset. Data augmentation technique and dropout is also used in the experiment during the process of building the network model to reduce the problem of overfitting. When there is a small number of samples for the model to learn from, the model will be unable to generalize to new and unseen data. Hence, data augmentation technique is used to generate more samples for the model to learn from. It works by augmenting the existing training samples via a few random transformations. The main purpose of data augmentation is to ensure the model will not see the same picture twice during training time and expose the model to much more aspects of data and thus generalize better.

In the Figure 2.13.[7] below, input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filter, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv.

layers are followed by max-pooling). Max-pooling is performed over a  $2 \times 2$ -pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.

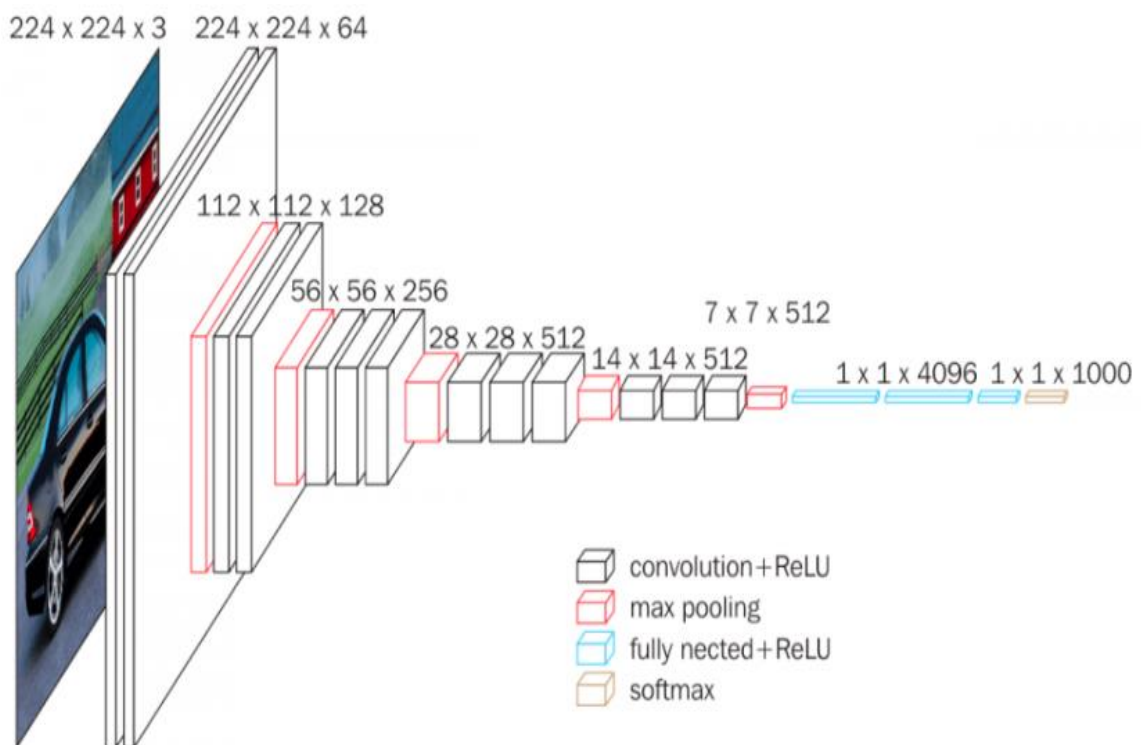


Figure 2.13. Convolutional Neural Network Architecture

The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple  $3 \times 3$  kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

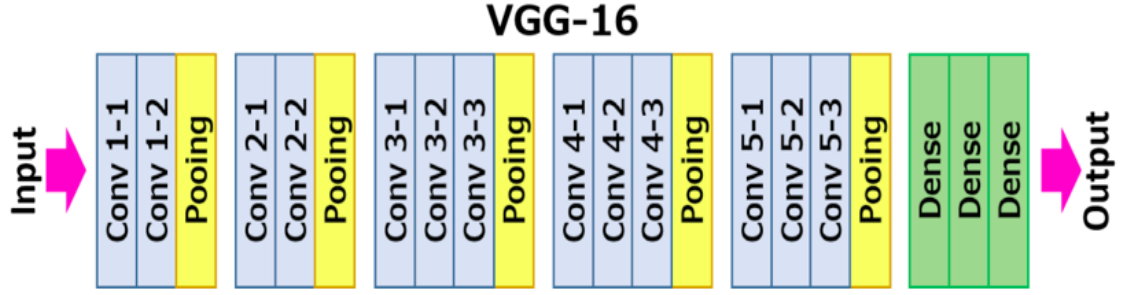


Figure 2.14. VGG16 Architecture

ImageNet[7] is a dataset of over 15 million labelled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labelled by human labellers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. ImageNet consists of variable-resolution images. Therefore, the images have been down-sampled to a fixed resolution of  $256 \times 256$ . Given a rectangular image, the image is rescaled and cropped out the central  $256 \times 256$  patch from the resulting image.

Concluding this paper, the author used VGG16 and transfer learning method to classify plant images, firstly by converting the images to smaller pixels according to the prerequisites of the VGG16 architecture, and the model achieves an accuracy rate of 92.7% in ImageNet having 14 million images in the dataset with 1000 classes.

## 2.7. Convolutional Neural Networks for Image Denoising in Infocommunication Systems

Noises reduce image quality and can lead to erroneous interpretation of useful information. Noisy images are difficult to analyze both programmatically and by humans. Image filtering algorithms are often used to reduce the effect of noise on images transmitted in infocommunication systems.

Traditional statistical image filtering algorithms are not always effective for the random nature of the noise spectrum. Hence, this paper talks about using convolutional neural networks for image denoising. Unlike traditional algorithms, denoising convolutional neural networks have architectural features that allow them to perform effective image filtering with unknown noise level. The authors of this paper have proposed to use denoising convolutional neural networks to generate a correction signal in the infocommunication system [8]. If the images transmitted in the infocommunication system have non-Gaussian noise, traditional methods do not provide a satisfactory quality of filtration. Image denoising was developed and implemented in TensorFlow convolutional neural network with special architecture, known as -DnCNN [10].

DnCNN uses ReLU (Rectified Linear Unit) activation functions after convolution. A subsampling operation was performed by the authors using pooling layers, reducing the dimensions of the generated feature matrices. In the proposed model, a group of pixels was compacted to one pixel, passing a nonlinear transformation. After several passages of image convolution, pooling remains a large set of ‘channels’ storing a small amount of data, which are interpreted as the most abstract concepts revealed from the original image. This data was then combined and transferred to an ordinary, fully-connected layer. DnCNN predicts the residual (difference between noisy and clean) image accurately. It removes the latent clean image with operations in the hidden layers. Thus, DnCNN can be used to generate a correction signal in an infocommunication system that transmits a noisy image.

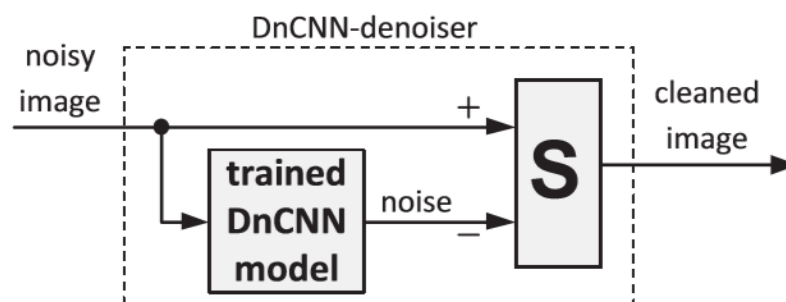


Figure 2.15. Image denoising in infocommunication system



Before using the model in DnCNN denoiser, the following operations are performed:

- preparation of training and test data (a set of clean and noisy images)
- data vectorization and tensors preparation
- Model training for several epochs, on each of which the neural network receives all data from the training set, the quality of the model is testing after each epoch.

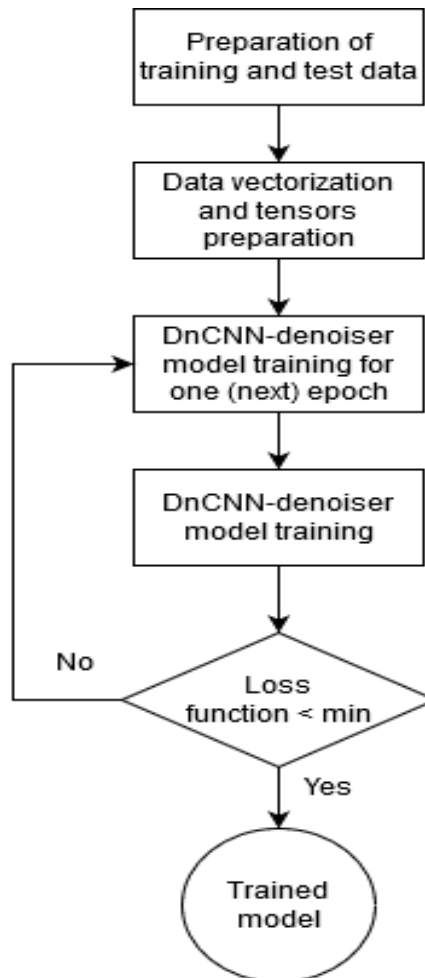


Figure 2.16. Algorithm for DnCNN model training

The novelty of the proposed infocommunication system is the use of a DnCNN-denoiser, which gets a signal from the receiver for filtering. DnCNN-denoiser has the capacity to handle the blind Gaussian denoising with unknown noise level and can be easily retrained if necessary.

In some cases, even the nature of the noise may initially be unknown. The advantage of using CNN for image denoising is the ability to filter transmitted images in real time. The model is considered correctly trained when the loss function reaches a certain minimum value. The choice of different loss functions gives a different training quality. In conclusion, a single DnCNN model can be used for well-known denoising tasks (Gaussian denoising, JPEG deblocking etc).

## **2.8. Classification of Breast Tumours in Mammograms using a Neural Network: Utilization of Selected Features**

The paper[9] proposed removal of nonuniform background trends superimposed on mammographic images, and two selected features were calculated - the standard deviation and entropy of an image, which represent the texture of the image. Instead of directly inputting the image to the neural network, representative features are presented to the network to learn and test. This helps in reducing computational overhead of training vis-à-vis early error control or inconsistency in dataset even after preprocessing of data.

An artificial neural network approach to classification of possible tumors into benign and malignant ones in mammograms was applied earlier by researchers, and an average of 75% recognition rate was shown. The authors of this paper[9] propose a different method for improving the recognition rate.

In earlier researches, image patterns were pre-processed first, to be used for learning and testing, by removing background trends and by reducing image matrix size and gray levels to 16x16x3 bits from the original extracted region of 256x256x10 bits around the tumor. Subsequently the pre-processed images were used as input pattern. Preliminary results showed that the neural network could correctly classify the benign and malignant tumors by an average of 75% recognition rate.

To further improve the recognition and classification rate, the authors have proposed an alternative method. Instead of directly inputting the image to the neural network, representative features are presented to the network to learn and test. However, determining a set of meaningful and representative features is usually a difficult problem.

Therefore, the two values – ‘Standard Deviation’ and ‘Entropy’ are considered as extracted features and then used as input to the neural network. 40 breast images consisting of 20 benign and 20 malignant tumor patterns were used. The size of each image was 1024x1024 pixels with 10-bit grey levels. After this, region of interest (ROI) (256x256 pixels) was cut from each original image. The ROI was considered to be an "extracted image" which was then used for further pre-processing and analysis. A total of 10 benign and 10 malignant tumor patterns were chosen randomly from the 40 image samples and used as training data, whereas the remaining patterns were used as testing data.

At the first stage, nonuniform background trend superimposed on the sample images were removed. This pre-processing is essential to eliminate the patterns of the underlying tumors from the sample images because the unwanted background trend may affect recognition. The standard deviation and entropy were then calculated and used as input data given to the neural network.

The results show that the recognition performance of the proposed neural network was achieved as 100% for various numbers of hidden units ranging from 30 to 70. The confidence level which is within the range of 0 - 1 is determined from the weighted values in the output layer. The larger the value, the higher the degree of possibility of correct classification.

In the next phase, the training and testing set was interchanged for cross-validation testing. A recognition of accuracy of 100% was also achieved. Conclusively, the experimental results show that the proposed method provides an improvement of 25% over the previous methods.

Number of hidden units	20	30	40	50	60	70	80
Recognition rate (%)	*	100	100	100	100	100	*
Confidence level		0.99	0.99	0.99	0.99	0.99	

Table 2.4. Recognition rates for various numbers of hidden units in the hidden layer

The authors also compared its performance with that of a statistical approach by employing a linear discrimination line obtained from regression lines to strengthen their confidence in the use of the new neural network. The superior performance of the neural network classifier was the final validation. This encouraging result indicates that this method may be useful for classification of benign and malignant tumors in mammograms. But the authors still wish to test the model and verify on a larger dataset as part of their future scope.

## 2.9. Summary of Literature Survey

Paper	Tools/Techniques	Objective
Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning	Convolutional Neural Network, Deep Learning, ReLU, SoftMax, tanh	Compared SoftMax, ReLU, tanh activation functions and found ReLU to be the most optimum activation function with highest accuracy.
Transfer Learning for Image Classification	Transfer Learning, AlexNet, VGG16, VGG19	Compared AlexNet, VGG16, and VGG19 models and found VGG19 to be the most optimum model with highest accuracy followed by VGG16.
Image Classification via Support Vector Machine	Support Vector Machine	Compared Neural Network and Support Vector Machine and found Support Vector Machine to be the most optimum with highest accuracy.
A KNN Research Paper Classification Based on Shared Nearest Neighbor	K-Nearest Neighbor	Paper presents a KNN text categorization method based on shared nearest neighbor, effectively combining the BM25 similarity calculation method and the Neighborhood Information of samples.
Credit Card Fraud Detection using Random Forest Algorithm	Random Forest Algorithm	Reviewed implementation of Random Forest Algorithm for increasing accuracy in detecting credit card frauds and its advantages over existing algorithms.
VGG16 for plant image classification with transfer learning and data augmentation	VGG16, Transfer Learning	Paper discusses the potential of VGG16 architecture model which is good for classification of plant images with a classification accuracy of 96.25% for Training and 89.96% for testing set with the help of transfer learning
Convolutional Neural Networks for Image Denoising in Infocommunication Systems	TensorFlow, Denoising Convolutional Neural Network	Detailed study to demonstrate the possibilities of using denoising convolutional neural networks to solve one of the most difficult tasks when performing the transfer of graphical information in infocommunication systems – denoising, using a DnCNN denoiser.
Classification of Breast Tumors in Mammograms using a Neural Network: Utilization of Selected Features	TensorFlow, Convolutional Neural Network	Study of proposed alternative algorithm for improving recognition rate by using representative features instead of presenting entire data-set at once.

Table 2.5. Summary of Literature Survey

## Chapter 3

### Design & Analysis

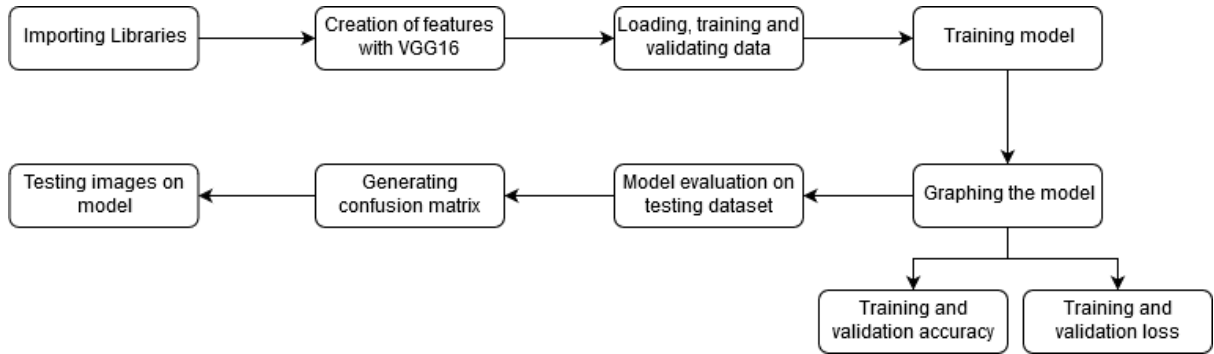


Figure 3.1. Design and analysis plan

Here, we present a methodology for the classification of fauna images, which will help ecologists and scientists to further study and/or improve habitat, environmental and extinction patterns. Figure 3.1. displays the proposed design of model for Fauna Image Classification using Convolutional Neural Network. The Animal-10 dataset[10] used for the classification is available open-source from Kaggle. We have used Convolutional Neural Network with Leaky ReLU activation function and VGG16 architecture for our model. The initial step taken by the system aims at creation of features with VGG16 model. Application of Image Processing along with Loading, Testing, Training, and Validating the dataset before the training step helps to remove the noise, obstacles, distortion and dirt from the images. The next step uses Convolutional Neural Network along with Leaky ReLU to train the model to accurately and precisely classify animal classes. In order to avoid the problem of Dying ReLU, where some ReLU neurons essentially die for all inputs and remain inactive no matter what input is supplied, here no gradient flows and if large number of dead neurons are there in a neural network its performance is affected. To resolve this issue, we make use of what is called Leaky ReLU, where slope is changed left of  $x=0$  and thus causing a leak and extending the range of ReLU. After training the model, we graph the model's training and validation accuracy and loss to have insights about how well the model is trained. Lesser the loss, more the accuracy. The next step is to generate classification matrix and confusion matrix to have exact details about how correctly the model is trained and classifying, as we cannot only rely on the accuracy. Lastly, we tested our model with sample data and found it to be accurately classified.

# Chapter 4

## Implementation

### 4.1. Importing Libraries

```
importing necessary libraries

In [1]: import pandas as pd
import numpy as np
import itertools
import keras
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras.models import Sequential
from keras import optimizers
from keras.preprocessing import image
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from keras.utils.np_utils import to_categorical
import math
import datetime
import time

Using TensorFlow backend.

In [3]: #Loading vgg16 model
vgg16 = applications.VGG16(include_top=False, weights='imagenet')

WARNING:tensorflow:From C:\Users\KAVISH\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

In [4]: datagen = ImageDataGenerator(rescale=1. / 255) #needed to create the bottleneck .npy files
```

Figure 4.1. Importing necessary libraries

We imported the required libraries for the neural network, such as Panda library, which is used for providing high-performance, easy-to-use data structures and data analysis, NumPy is used for mathematical and logical operations on arrays can be performed, Keras is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible, and many more. The images were resized, width=224 and height=224 as per VGG16 model requirements. A bottleneck file was created to encourage the network to compress feature representations to best fit in the available space, in order to get minimum loss during training. Bottleneck file system was used to convert all image pixels into their number (NumPy array) correspondent and store it in storage system. We simply tell our network where images are located in our storage so the machine knows where is what. We defined the epoch and batch sizes for our neural network. We categorized the image dataset into train, validate and test and loaded it into the network. We also loaded a pre-trained VGG16 model as it uses only 11 convolutional layers and pretty easy to work with.

## 4.2. Creation of features/weights with VGG16

```
Creation of weights/features with VGG16

In [5]: #this can take an hour and half to run so only run it once.
start = datetime.datetime.now()

generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_train_samples = len(generator.filenames)
num_classes = len(generator.class_indices)

predict_size_train = int(math.ceil(nb_train_samples / batch_size))

bottleneck_features_train = vgg16.predict_generator(generator, predict_size_train)

np.save('bottleneck_features_train.npy', bottleneck_features_train)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)

Found 13412 images belonging to 6 classes.
Time:  1:35:04.754841

In [5]: start = datetime.datetime.now()
generator = datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_validation_samples = len(generator.filenames)

predict_size_validation = int(math.ceil(nb_validation_samples / batch_size))

bottleneck_features_validation = vgg16.predict_generator(
    generator, predict_size_validation)

np.save('bottleneck_features_validation.npy', bottleneck_features_validation)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)

Found 2549 images belonging to 6 classes.
Time:  0:22:26.636826

In [5]: start = datetime.datetime.now()
generator = datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_test_samples = len(generator.filenames)
predict_size_test = int(math.ceil(nb_test_samples / batch_size))
bottleneck_features_test = vgg16.predict_generator(generator, predict_size_test)
np.save('bottleneck_features_test.npy', bottleneck_features_test)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)

Found 1845 images belonging to 6 classes.
Time:  0:13:43.012092
```

Figure 4.2. Creating weights/features with VGG16

We created weights/features with VGG16 model to finetune the neural network to perform well with the input data. Once the files have been converted and saved to the bottleneck file, we loaded and prepared them for our convolutional neural network. This was also a good way to make sure all our data have been loaded into bottleneck file. Data quality is vital to ensure accuracy and reliability. The train network successfully found 13412 train image which belonged to 6 animal classes in 95.04 minutes, the validate network successfully found 2549 validated images which belonged to 6 animal classes in 22.26 minutes, and the test network successfully found 1845 test images which belong to 6 animal classes in 13.43 minutes.

## 4.3. Loading, training, and validating data

```
Loading training, validation and testing data

In [6]: #training data
generator_top = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)
nb_train_samples = len(generator_top.filesnames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
train_data = np.load('bottleneck_features_train.npy')

# get the class labels for the training data, in the original order
train_labels = generator_top.classes

# convert the training labels to categorical vectors
train_labels = to_categorical(train_labels, num_classes=num_classes)

Found 13412 images belonging to 6 classes.

In [7]: #validation data
generator_top = datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_validation_samples = len(generator_top.filesnames)

validation_data = np.load('bottleneck_features_validation.npy')

validation_labels = generator_top.classes
validation_labels = to_categorical(validation_labels, num_classes=num_classes)

Found 2549 images belonging to 6 classes.

In [8]: #testing data
generator_top = datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_test_samples = len(generator_top.filesnames)

test_data = np.load('bottleneck_features_test.npy')

test_labels = generator_top.classes
test_labels = to_categorical(test_labels, num_classes=num_classes)

Found 1845 images belonging to 6 classes.
```

Figure 4.3. Loading, training, and validating data

This module loads training and validation data for testing the model. The training network successfully found 13412 train images which belonged to 6 animal classes, and the validation network successfully found 2549 validated images which belonged to 6 animal classes. It loads testing data for testing the model. The training network successfully found 1846 train images which belonged to 6 animal classes



## 4.4. Training model

Training of model

```
In [9]: start = datetime.datetime.now()
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.5))
model.add(Dense(50, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])

history = model.fit(train_data, train_labels,
                  epochs=7,
                  batch_size=batch_size,
                  validation_data=(validation_data, validation_labels))

model.save_weights(top_model_weights_path)

(eval_loss, eval_accuracy) = model.evaluate(
    validation_data, validation_labels, batch_size=batch_size, verbose=1)

print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
print("[INFO] Loss: {}".format(eval_loss))
end = datetime.datetime.now()
elapsed = end - start
print('Time: ', elapsed)
```

WARNING:tensorflow:From C:\Users\KAVISH\Anaconda3\lib\site-packages\tensorflow\python\ops\math\_ops.py:3066: to\_int32 (from tensorflow.python.ops.math\_ops) is deprecated and will be removed in a future version.  
Instructions for updating:  
Use tf.cast instead.  
Train on 13412 samples, validate on 2549 samples  
Epoch 1/7  
13412/13412 [=====] - 13s 976us/step - loss: 0.7251 - acc: 0.7372 - val\_loss: 0.4256 - val\_acc: 0.8474  
Epoch 2/7  
13412/13412 [=====] - 10s 728us/step - loss: 0.4087 - acc: 0.8612 - val\_loss: 0.3454 - val\_acc: 0.8858  
Epoch 3/7  
13412/13412 [=====] - 9s 708us/step - loss: 0.3293 - acc: 0.8879 - val\_loss: 0.3806 - val\_acc: 0.8674  
Epoch 4/7  
13412/13412 [=====] - 10s 715us/step - loss: 0.2827 - acc: 0.9043 - val\_loss: 0.2674 - val\_acc: 0.9157  
Epoch 5/7  
13412/13412 [=====] - 10s 756us/step - loss: 0.2416 - acc: 0.9203 - val\_loss: 0.2763 - val\_acc: 0.9109  
Epoch 6/7  
13412/13412 [=====] - 10s 737us/step - loss: 0.2036 - acc: 0.9295 - val\_loss: 0.3774 - val\_acc: 0.8847  
Epoch 7/7  
13412/13412 [=====] - 10s 756us/step - loss: 0.1822 - acc: 0.9409 - val\_loss: 0.2707 - val\_acc: 0.9184  
2549/2549 [=====] - 0s 181us/step  
[INFO] accuracy: 91.84%  
[INFO] Loss: 0.27072709791651656  
Time: 0:01:15.003557

```
In [10]: #Model summary
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 100)	2508900
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
dropout_2 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 6)	306

Total params: 2,514,256  
Trainable params: 2,514,256  
Non-trainable params: 0

Figure 4.4. Training model and model summary

We trained the neural model. Training a neural network involves finding a set of weights to best map inputs to outputs. Loss function is a function that tells us, how good our neural network is for a certain task. The problem of training is equivalent to the problem of minimizing the loss function. The procedure used to carry out the learning process in a neural network is called the optimization algorithm (or optimizer).

First step is to initialize the model with *Sequential()*. After that we flatten our data and add additional 3 hidden layers. Then, after we have created and compiled our model, we fit our training and validation data to it. Finally, we create an evaluation step, to check for the accuracy of our model training set versus validation set.

An epoch describes the number of times the algorithm sees the entire data set. So, each time the algorithm has seen all samples in the dataset, an epoch has completed. Neural network training algorithms involve making multiple presentations of the entire data set to the neural network. One forward pass and one backward pass of all the training examples. Each epoch must finish all batch before moving to the next epoch. A single presentation of the entire data set is referred to as an "epoch". 7 epochs resulting in 91.84% accuracy and loss of 0.2752.

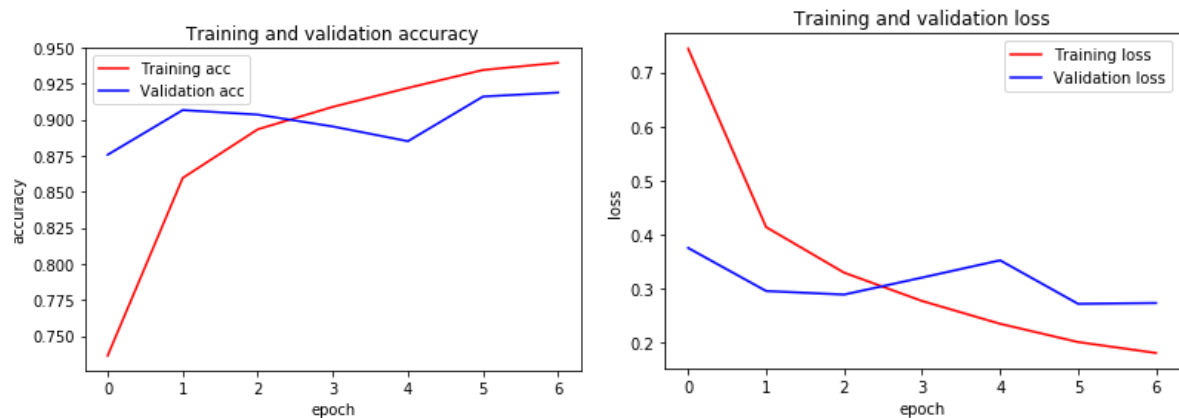
Keras provides a way to summarize a model. The summary can be created by calling the *summary()* function on the model that returns a string that in turn can be printed. The summary is textual and includes information about:

- The layers and their order in the model.
- The output shape of each layer.
- The number of parameters (weights) in each layer.
- The total number of parameters (weights) in the model.

In between the convolutional layer and the fully connected layer, there is a Flatten layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier. A dense layer represents a matrix vector multiplication. The values in the matrix are the trainable parameters which get updated during backpropagation. Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.

## 4.5. Graphing training and validation accuracy and loss

```
In [10]: #Graphing our training and validation
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```



### Model Evaluation on Testing Set

```
In [11]: model.evaluate(test_data, test_labels)
1845/1845 [=====] - 0s 186us/step
Out[11]: [0.2178296768083805, 0.9333333333979454]
```

Figure 4.5. Graphing training and validation accuracy and loss

We are graphing the training and validation accuracy and loss for each epoch. During an epoch, the loss function is calculated across every data item and it is guaranteed to give the quantitative loss measure at the given epoch and plotting curve across iterations only gives the loss on a subset of the entire dataset. The model evaluate function tells how well the machine we just made can predict against unseen data. Model evaluation states how well the machine we just made can predict against unseen data. There are two great methods to see how well our machine can predict or classify. One of them is the classification metrics and the other is the confusion matrix.

## 4.6. Model evaluation

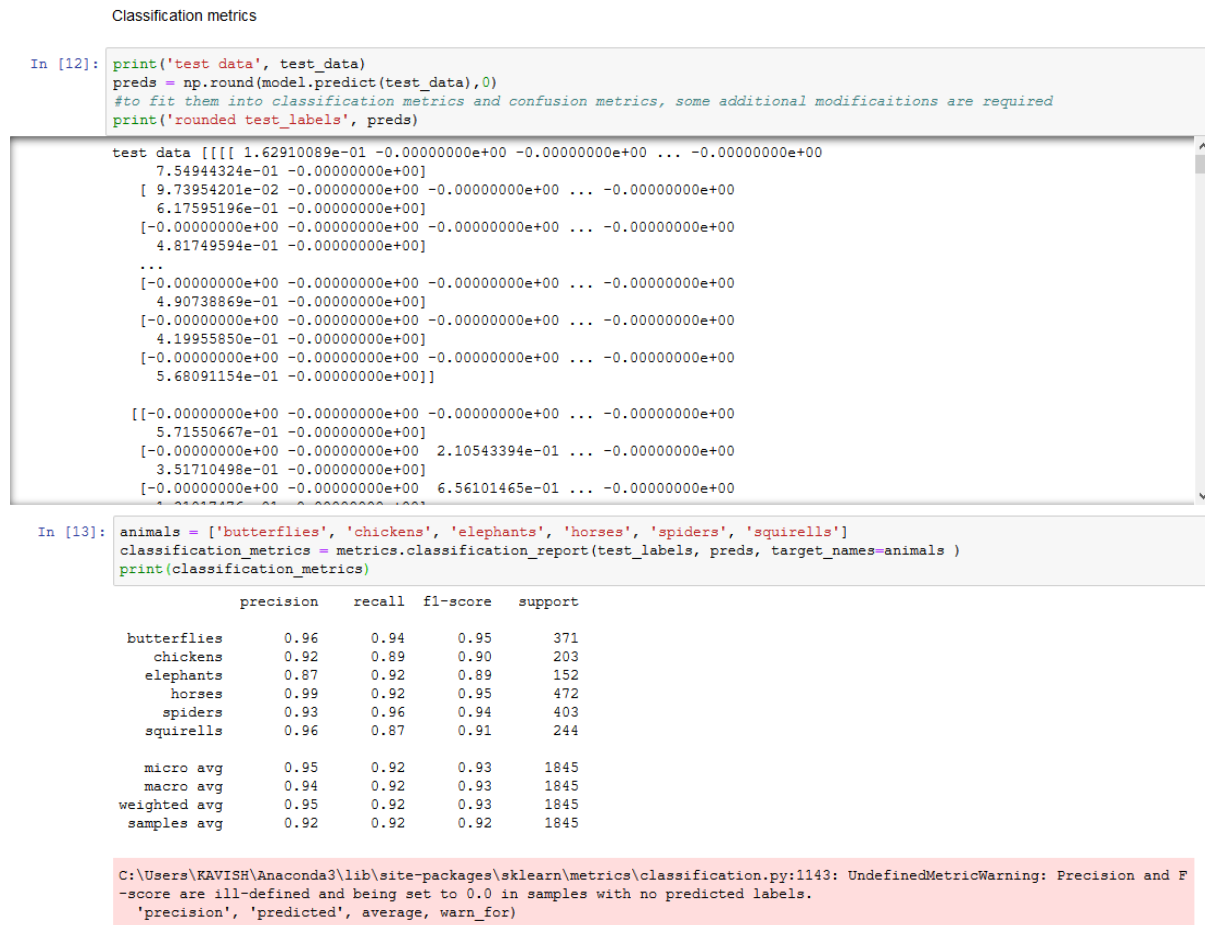


Figure 4.6. Classification metrics

We implemented and displayed Classification Metrics (precision, recall, F1-score, support), along with their micro and macro average and weighted and samples average. To use classification metrics, we had to convert our testing data into a different NumPy format, NumPy array, to read. Precision is good measure when classification accuracy is not a good indicator of your model performance, when your class distribution is imbalanced. Recall is defined as the fraction of samples from a class which are correctly predicted by the model. One popular metric which combines precision and recall is called F1-score. Support is the number of samples of the true response that lie in that class.

- Precision = True\_Positive / (True\_Positive + False\_Positive)
- Recall = True\_Positive / (True\_Positive + False\_Negative)
- F1-score = 2 \* Precision \* Recall / (Precision + Recall)

The higher the score the better the model is. To use classification metrics, we had to convert our testing data into a different NumPy format, NumPy array, to read. It then converted the code and executed it through the built-in classification metrics to give us a neat result.

## 4.7. Generating confusion matrix

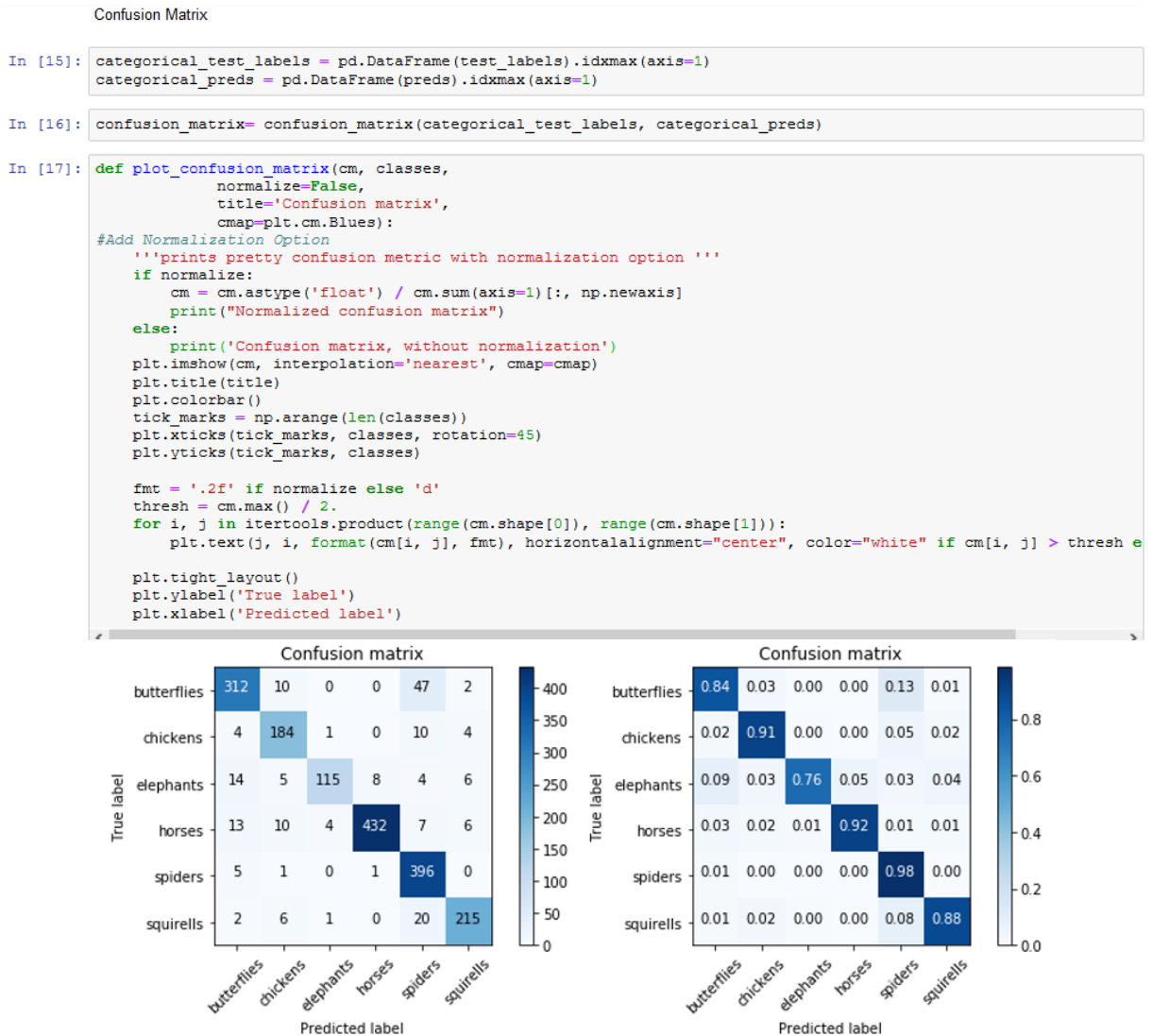


Figure 4.7. Generating Confusion Matrix

We implemented confusion matrix as we cannot only rely on the accuracy. Confusion matrix is used to evaluate the quality of the output of a classifier. Confusion matrix is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents instances in a predicted class and each column represents the instances in an actual class. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions. The NumPy array we created before is placed inside a data frame. Confusion matrix works best on data frames. The figures show the confusion matrix with and without normalization by class support size (number of elements in each class). This kind of normalization can be interesting in case of class imbalance to have a more visual interpretation of which class is being misclassified.

## 4.8. Testing images on model

Testing images on model

```
In [19]: def read_image(file_path):
        print("[INFO] loading and preprocessing image...")
        image = load_img(file_path, target_size=(224, 224))
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)
        image /= 255.
        return image
```

```
In [20]: def test_single_image(path):
        animals = ['butterflies', 'chickens', 'elephants', 'horses', 'spiders', 'squirrels']
        images = read_image(path)
        time.sleep(.5)
        bt_prediction = vgg16.predict(images)
        preds = model.predict_proba(bt_prediction)
        for idx, animal, x in zip(range(0,6), animals, preds[0]):
            print("ID: {}, Label: {} {}".format(idx, animal, round(x*100,2) ))
        print('Final Decision:')
        time.sleep(.5)
        for x in range(3):
            print('.'*(x+1))
            time.sleep(.2)
        class_predicted = model.predict_classes(bt_prediction)
        class_dictionary = generator_top.class_indices
        inv_map = {v: k for k, v in class_dictionary.items()}
        print("ID: {}, Label: {}".format(class_predicted[0], inv_map[class_predicted[0]]))
        return load_img(path)
```

```
In [27]: #path of image that is used for testing
        path = 'data/test/testimage.jpeg'
```




<pre>In [28]: test_single_image(path)  [INFO] loading and preprocessing image... ID: 0, Label: butterflies 0.0% ID: 1, Label: chickens 0.0% ID: 2, Label: elephants 99.89% ID: 3, Label: horses 0.1% ID: 4, Label: spiders 0.0% ID: 5, Label: squirrels 0.01% Final Decision: . . . . . ID: 2, Label: elephants</pre> <p>Out[28]:</p> 	<pre>In [24]: test_single_image(path)  [INFO] loading and preprocessing image... ID: 0, Label: butterflies 99.99% ID: 1, Label: chickens 0.0% ID: 2, Label: elephants 0.0% ID: 3, Label: horses 0.0% ID: 4, Label: spiders 0.01% ID: 5, Label: squirrels 0.0% Final Decision: . . . . . ID: 0, Label: butterflies</pre> <p>Out[24]:</p> 	<pre>In [32]: test_single_image(path)  [INFO] loading and preprocessing image... ID: 0, Label: butterflies 0.0% ID: 1, Label: chickens 0.0% ID: 2, Label: elephants 0.0% ID: 3, Label: horses 0.0% ID: 4, Label: spiders 0.0% ID: 5, Label: squirrels 100.0% Final Decision: . . . . . ID: 5, Label: squirrels</pre> <p>Out[32]:</p> 
---	---	--

Figure 4.8. Sample image output with accuracy and animal class

Finally, the last stage is the testing of the trained model on a sample image to check whether the neural network is trained accurately and is working error free. The image is fed in the neural network model and the model accurately classifies the animal class. The *def read\_image* function is letting our machine know that it has to load the image, change the size and convert it to an array. The *def test\_single\_image* function is using transfer learning's prediction model and an iterative function to help predict the image properly. The *path* is where we define the image location and finally the *test\_single\_image* cell block will print out the final result, depending on the prediction from the second cell block. Our neural network correctly identified different animal image with different background, noise, etc and classified it to the animal class with 99.89% accuracy.

## Chapter 5

### Results & Discussion

The proposed model was coded in Python and tested in Jupyter Notebook on the Animal-10 dataset[10] which contains 26,179 images from 10 animal classes. The model could achieve an accuracy of 91.84% for 6 animal classes. The neural network could successfully identify the animal image and classified it to the correct animal class with an accuracy of 99.89%. The model successfully detected 1845 testing images from 6 animal classes, 13412 training images from 6 animal classes, and 2549 validated images from 6 animal classes. Table 5.1. provides insights into the number of images for each animal classes.

<b>Animal</b>	<b>Number of Images</b>
Butterflies	2112
Chickens	3098
Elephants	1446
Horses	2623
Spiders	4821
Squirrels	1862
<b>Total</b>	<b>15,962</b>

Table 5.1. Animal classes in the dataset

Different animal image with different background, noise, distortion, unclear images, half images, etc are successfully classified into their specific animal class with accuracy. Processing such a large volume of images and videos captured from camera traps manually is extremely expensive, time-consuming and also monotonous. This presents a major obstacle to scientists and ecologists to monitor wildlife in an open environment. The neural network we developed, in turn, can therefore speed up research findings, can also lead to discovery of potential new habitats as well as new, unseen and/or rare species of animals (on the verge of extinction) - within the same class, construct more efficient monitoring systems and subsequent management decisions, having the potential to make significant impact to the world of ecology and trap camera images analysis. The technical applications include: Automated Image Organization, Stock Photography and Video Websites, Visual Search for Improved Discoverability, Image Classification for Websites with Large Visual Databases, Image and Face Recognition on Social Networks.

## Chapter 6

### Conclusion & Future Work

The neural network model could achieve an accuracy of 91.84% for 6 animal classes. The neural network could successfully identify the animal image and classified it to the correct animal class with an accuracy of 99.89%. The outcome of the research work developed for fauna image classification using convolutional neural network is summarized. It addresses the implementation of convolutional neural network for fauna image classification. The performance of ReLU activation function was found to be most accurate. We found VGG16 model to be most accurate and appropriate for image classification. The neural network is trained to classify image of an animal and help identify animal class with accuracy. We have trained our neural network in such a way that it can train new animal class by simply feeding the neural network with labelled images. Concluding, the proposed fauna image classification using convolutional neural network can be used extensively for fauna image classification.

#### 6.1. Future work

- Developing a UI for the project for easy use for ecologist, photographers, computer researchers
- Improvising the classification accuracy, precision and reduction in terms of error, training and testing time
- The image classification model can be enhanced in future, by including more low-level features such as shape and spatial location features apart from optimizing the weights and learning rate of the neural network.

When these enhancements are incorporated in the classification system, it would help further improve the performance and be useful for applications meant for the explicit classification system.



## REFERENCES

- [1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. “Going deeper with convolutions”, IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9
- [2] Dhruv Rathi, Sushant Jain, Dr. S. Indu, “Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning”, International Conference of Advances in Pattern Recognition, 2017
- [3] Manali Shaha, Meenakshi Pawar, “Transfer learning for image classification”, 2<sup>nd</sup> International conference on Electronics, Communication and Aerospace Technology (ICECA), 2018
- [4] Xiaowu Sun, Lizhen Liu, Hanshi Wang, Wei Song, Jingli Lu, “Image Classification via Support Vector Machine”, 4th International Conference on Computer Science and Network Technology (ICCSNT), 2015
- [5] Yun-lei Cai, Duo Ji, Dong-feng Cai, “A KNN Research Paper Classification Method Based on Shared Nearest Neighbor”, NTCIR-8 Workshop Meeting, Tokyo, Japan, 2010
- [6] M. Suresh Kumar, V. Soundarya, S. Kavitha, E.S. Keerthika, E. Ashwini, “Credit Card Fraud Detection Using Random Forest Algorithm”, 3rd International Conference on Computing and Communications Technologies (ICCCT), 2019
- [7] Mohamad Aqib Haqmi Abas, Nurlaila Ismail, Ahmad Ishan Mohd Yassin, Mohd Nasir Taib, “VGG16 for plant image classification with transfer learning and data augmentation”, International Journal of Engineering & Technology, 2018
- [8] Oleksii Sheremet, Kateryna Shernemet, Oleksander Sadovoi, Yuliia Sokhina, “Convolutional Neural Networks for Image Denoising in Infocommunication Systems”, International Scientific-Practical Conference (Problems of Infocommunications, Science and Technology-PICS&T), 2018
- [9] H. Fujita, K. Horita, T. Endo, C. Kido, T. Ishigaki, “Classification of Breast Tumors in Mammograms using a Neural Network: Utilization of Selected Features”, International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), 1993
- [10] Kaggle.com, Animals-10 dataset [Online] Available at: <https://www.kaggle.com/alessiocrrado99/animals10>

## PUBLICATIONS

Title of Paper	A Survey on Image Classification Techniques
Author Names	Kavish Sanghvi, Adwait Aralkar, Saurabh Sanghvi, Ishani Saha
Publishing House	Hosting by Elsevier SSRN. Peer review under responsibility of International Conference on Sustainable Computing in Science, Technology & Management (SUSCOM-2020)
Impact Factor (if any)	-
Indexing (if any)	-

Title of Paper	Fauna Image Classification using Convolutional Neural Network
Author Names	Kavish Sanghvi, Adwait Aralkar, Saurabh Sanghvi, Ishani Saha
Publishing House	Science and Engineering Research Support Society (SERSC). International Conference on Computing Technologies for transforming the Automated World (ICCTAW-2020)
Impact Factor (if any)	-
Indexing (if any)	Web of Science

## **ACKNOWLEDGEMENT**

We would like to express our deepest appreciation to all those who provided us the possibility to achieve the project goals. A special gratitude to our final year project mentor, Prof. Ishani Saha, who has invested her full effort in guiding the team in achieving the goal, contributed in stimulating suggestions and encouragement, and helped us to coordinate the project.

Furthermore, we would also like to appreciate the guidance given by the panels especially in our project presentation that has improved our project with their valuable comments and advices. The constant association with the members of the Department of Computer Engineering has been most pleasurable, without their help and counsel, always generously and unstintingly given, the completion of this work would have been immeasurably more difficult.