STUDY OF SCALABLE DEEP NEURAL NETWORK FOR WILDLIFE ANIMAL RECOGNITION AND IDENTIFICATION

A Thesis presented to the Department of

Computer Science and Engineering

African University of Science and Technology

Abuja, Nigeria.

In partial fulfillment of the Requirements for the Degree of

Masters of Science in Computer science

By

Yohanna Yerima Williams

June 2019.

African University of Science and Technology [AUST]

Knowledge is Freedom

APPROVAL BY

Supervisor

Surname: BEN ABDALLAH

First name: ABDERAZEK

Signature 

The Head of Department

Surname: DAVID

First name: Amos

Signature: 

CERTIFICATION

This is to certify that the thesis titled “Architecture and design of scalable deep neural network for wild life animal recognition and identification” submitted to the department of Computer Science and Engineering African University of science and technology Abuja, Nigeria for the award of Master’s Degree is a record of original research carried out by Yohanna Yerima Williams in the department of Computer Science and Engineering.

SIGNATURE PAGE

STUDY OF SCALABLE DEEP NEURAL NETWORK FOR WILDLIFE ANIMAL RECOGNITION AND IDENTIFICATION

By

Yohanna Yerima Williams

A THESIS APPROVED BY THE DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RECOMMENDED:

Supervisor, Prof Ben Abdallah

Head of Department, Prof David Amos

APPROVED BY:

Chief Academic Officer

Date

ABSTRACT

Recently, deep learning techniques have been used significantly for large scale image classification targeting wildlife prediction. This research adopted a deep convolutional neural network (CNN) and proposed a deep scalable CNN. Our research essentially modifies the network layers (scalability) dynamically in a multitasking system and enables real-time operations with minimum performance loss. It suggests a straightforward technique to access the performance gains of the network while enlarging the network layers. This is helpful as it reduces redundancy in network layers and boosts network efficiency. The architecture implementation was done in software using keras framework and tensorflow as the backend on the CPU and to corroborate the universality and robustness of our proposed approach; we train our model on a GPU with a newly created dataset named “Zedataset”, preprocessed for performance evaluation. Results obtained from our experimentations show that our proposed architecture design will perform better with more dataset at the set optimum parameters.

Keywords: GPU, keras, deep CNN, CNN, Scalability, tensorflow, image classification, optimum parameters, backend.

DEDICATION

I dedicate this thesis to Almighty God who has been there for me from the start to the finish.

ACKNOWLEDGMENTS

I am grateful to God Almighty for bringing me to this level in my career and to my family members for their continuous support throughout this work.

Most of all I would like to thank my supervisor Prof Ben Abdallah for his guidance, support, suggesting corrections and proofreading throughout my work.

Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

K-NN	K-Nearest Neighbour
LR	Linear Regression
MLP	Multi-Layer Perceptron
NB	Naïve Bayesian
Pred.	Prediction
RL	Reinforcement Learning (RL)
RLFM	Regression based latent factors (RLFM)
RNNLM	Recurrent Neural Network Language Model (RNNLM)
ROC	Receiver Operating Characteristic (ROC)
ReLU	Rectified Linear Unit (ReLU)
SBM	Stochastic block model (SBM)
SBO	Structured Bayesian optimization (SBO)
SBSE	Search-based software engineering (SBSE)
SCH	Stochastic convex hull (SCH)
SGD	Stochastic Gradient Descent (SGD)
SGVB	Stochastic Gradient Variational Bayes (SGVB)
CNN	Convolutional Neural Networks (CNN)
RL	Reinforcement Learning

ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
CV	Computer Vision
DNN	Deep Neural Network

Chapter One

Background of the study

1.0 Introduction

The task of identifying and recognition of animals from photos has long been standing as there is no unique method that provides a robust and efficient solution to all situations. Several researchers used long-standing traditional approaches for its implementation with the problem still hanging in limbo as the task hugely involve collecting a large volume of images which predominantly is conducted manually with possibly images having an imperfect quality which sometimes affect the speed of classification, accuracy even for domain experts. More so, processing these image sets is time-consuming, effort demanding, and comes at a very high cost as it is an overwhelming amount of data that is collected.

In recent years, much attention has focused on using deep neural network based techniques in the area of image processing, particularly animal recognition and identification. However, the increase in the performance characteristics of the network depends on how scalable the network is designed. In machine learning, scalability is often defined as the result that even the slightest change in the size of the network parameters such as the network layers, training sets has on the computational performance of an algorithm (accuracy, memory allocation, speed of processing). So the question is to find a balance or in order words getting a suitable solution quick off the mark and most effectively. This is of serious concern as in scathing circumstances where the existence of temporal or contiguous constraints like real-time applications

dealing with large datasets, unapproachable computational problems demanding learning or first prototyping needing quickly implemented the result.

To deal with a large dataset, it is expedient to minimize the training time and allotted memory space while preserving accuracy; however, till date, most proposed deep learning algorithms do not proffer a proper trade-off among them. To contain these issues above, we aim to optimize floating points by changing them to fixing points to reduce memory complexity and yield faster processing in the network. In this research, the convolutional neural network framework will be used for animal identification and prediction, while stochastic gradient descent is used to optimize the parameters (i.e., weights, biases) of the network through error backpropagation with momentum and adaptive learning rate. Network layers and nodes in each hidden layer will be added in systematic experimentation and intuition with a robust test to harness.

1.1 Concept of Deep learning

Deep learning is an offshoot of machine learning, which is not new to the field of informatics and predictive analysis. However, recently, it has drawn much attention as neuroscientist, psychologist, engineers, economist, AI workers attempt to explore their learning potential. Deep learning approaches are a set of algorithms that strive to model data with extreme abstractions using a replica architecture with tortuous formation. It is one among the many segments of machine learning techniques based on the concept of learning representations of raw data which could be in a way such as the intensity per pixel value of a data or sections of a specific figure in a more abstract way.

There are several numbers of ways the area of deep learning has been represented as it is a subset of machine learning techniques that

i. Uses multiple layers with nonlinear processing units cascaded for feature extraction
ii. Are based on the (unsupervised) learning multiple data representations where hierarchical representation is formed when higher-level features are derived from lower level features.
iii. Learned multiple levels of representations corresponds to different levels of abstraction.

1.2 Definition of learning

One challenging fact when setting up the objectives of deep learning is the definition of learning. Learning is rather conceptual and as to those who have made efforts to give it meaning (psychologists, philosophers, etc.) have only succeeded in uncovering one among the many faces of the complex procedure.

However, there are some views of learning which has been acceding to mostly by those who have made continuous efforts to divulge the concept, and these on many occasion provides reasonable interpretation of the process. Some are the following:

i. There exist a system manipulating information provided by its environment and is capable of improving its self.
ii. The system has numerous ways of altering its current state and information provided can usually take many forms.
iii. The system is capable of remembering and recalling things that it has experienced.

1.3 Concept of scalability in machine learning

Scalability has increasingly been integrated over the years as part of deep learning. This is as a result of the likelihood of performance characteristics been affected as recently; most deep neural networks are hugely involved with the overwhelming size of the dataset.

Scalability, as defined in machine learning, is the effect that a change in system parameters has on the performance characteristics of an algorithm. Its methods could be like increase in the number of nodes, network layers, and hidden layers by systematic experimentation and/or intuition. This is done to ensure faster processing with huge dataset while preserving some performance characteristics like (accuracy, memory allocation) and reduction in the network complexity.

1.4 Problem statement

There has been a rise in cases of human-animal attacks and human-vehicle collision with the latter been prevalent in Nigeria. There are about 500-1000 vehicle collisions with large animals each year that result in more than 1 billion Naira in damages. Source (Federal road safety annual report, 2017).

To cope with this problem, machine learning based techniques could be employed, which may be on CCTV cameras connected to the relevant response team for surveillance of animals in both remote and urban places to save lives.

1.5 Aim of the research

The aim of this thesis is to provide a scalable, suitable, more generic and optimized network capable of processing huge amount of dataset even with images having an imperfect quality or varied deformations in real time while preserving better test accuracy.

1.6 Objectives of the research

Having at hand the different views of people as regards to what learning seems to be and how to attain it. One can perceive how challenging it is to interpret deep learning and even to set out some clear objectives. Although the concept of learning has cleared the air despite that the approach to deep learning by different people differs. The aim of this research is as follows:

i. Develop an artificial learning system capable of being adaptive and self-improving
ii. Develop a neural network with optimized parameters whose computational performance is unaffected by scalability.
iii. Develop a neural network system architecture with reduced complexity for large scale image classification or prediction.

1.7 Structure of the research

Chapter 1 presents a brief introduction of the research concept primarily deep learning, objectives of the project, and the aims.

Chapter 2 presents supporting theories of the research concept following brief introduction of deep learning concept and learning, forming a link with a classification problem, then give a brief account of the different classification approaches ranging from statistical methods to genetic algorithms. Two best learning approaches will be examined and finally, a brief account of similar works done will follow.

Chapter 3 will presents the theoretical analysis of the adopted algorithm with the proposed layers. The following information is provided:

i. A detailed description of the algorithm focusing on its peculiarities

ii. The design of the algorithm with a detailed explanation of its layers.

Chapter 4 will describes the experiments and presents the results which will be statistically analyzed to check for relative performance and the validation of the theoretical estimates presented in the previous chapter.

Chapter 5 summarises the results presented in the thesis and concludes their importance in the context of recognition and identification.

Chapter Two

Literature review

2.0 Introduction

In this section, we present supporting theories of the research concept following brief introduction of deep learning concept and learning, forming a link with a classification problem, then give a brief account of the differing digital image classification approaches ranging from per pixel classification to object-oriented classification. Two best classification approaches will be examined, and finally, a brief account of similar works done will follow.

2.1 Basic Concept and Terminology

Machine learning is a branch of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the construction and study of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs to make data-driven predictions or decisions rather than following procedural program instructions. Machine learning is most at times, often overlaps with computational statistics; a discipline that also specializes in prediction-making. It has strong ties to mathematical optimization, which deliver methods, theory, and application domains to the field. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms are infeasible. Example applications include spam filtering, optical character recognition (OCR), search engines, and computer vision. Machine learning is sometimes conflated with data mining, although that focuses more on exploratory data analysis. Machine learning

and pattern recognition “can be viewed as two facets of the same field.” (Machine Learning Wikipedia full guide, 2017)

2.2 Digital image classification

Image classification can be said to be a process of assigning all pixels in the image to particular classes or themes based on spectral information represented by the digital numbers (DNs). The classified image comprises a mosaic of pixels, each of which belongs to a particular theme and is a thematic map of the original image (Anupam Anand, 2018). The main steps of image classification as shown in figure 2.2 may include image pre-processing, feature extraction, training samples selection, selection of suitable classification approaches, post-classification processing, and assessment accuracy (黄正华, 2014). However, Classification will be executed on the base of spectral or spectrally defined features, such as density, texture, etc., in the feature space. It can be said that classification divides the feature space into several classes based on a decision rule (黄正华, 2014). There are basically two approaches to image classification, namely; per pixel image classification and object-oriented classification. Per pixel is the most commonly adopted method as the algorithm categorizes each input pixel into a spectral feature class based solely on its multispectral vector. No context or neighborhood evaluation is involved (Shrivastav & Singh, 2019) while in object-oriented classification, the input pixels are grouped into spectral features (objects features) using image segmentation. These objects are characterized in both the raster and vector domains. The objects are classified using both spectral and spatial cues (Shrivastav & Singh, 2019).

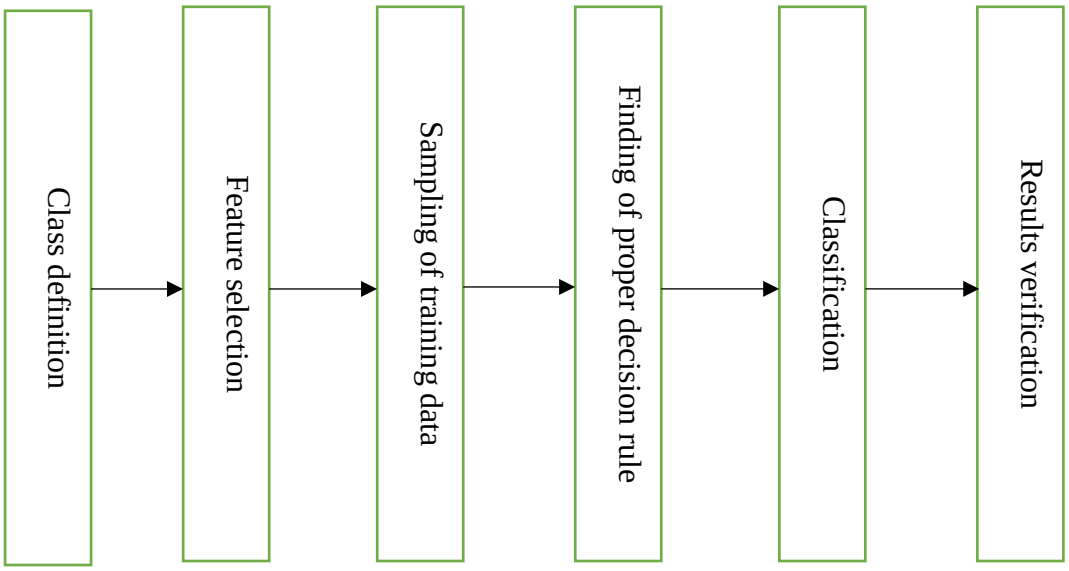


Figure 2. 1: Classification procedure

Two most common methods of per pixel approach are namely;

i. Supervised learning

ii. Unsupervised learning

2.2.1 Supervised learning

Supervised learning is a machine learning task of inferring a function from labeled training data. In order words for example, given a set of example pairs *(x, y), $x \in X$, $y \in Y$* and the aim is to find a function *f:* $X \rightarrow Y$ in the allowed class of functions that matches the examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and the desired output value (also called the supervisory signal). A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a “reasonable” way (Machine Learning Wikipedia full

guide, 2017). Some of the examples of supervised classification techniques are Back Propagation Network (BPN), Learning Vector Quantization (LVQ), Support Vector Machine (SVM), etc. Solving the problem of supervised learning requires the following steps as shown in figure 2.2.1;

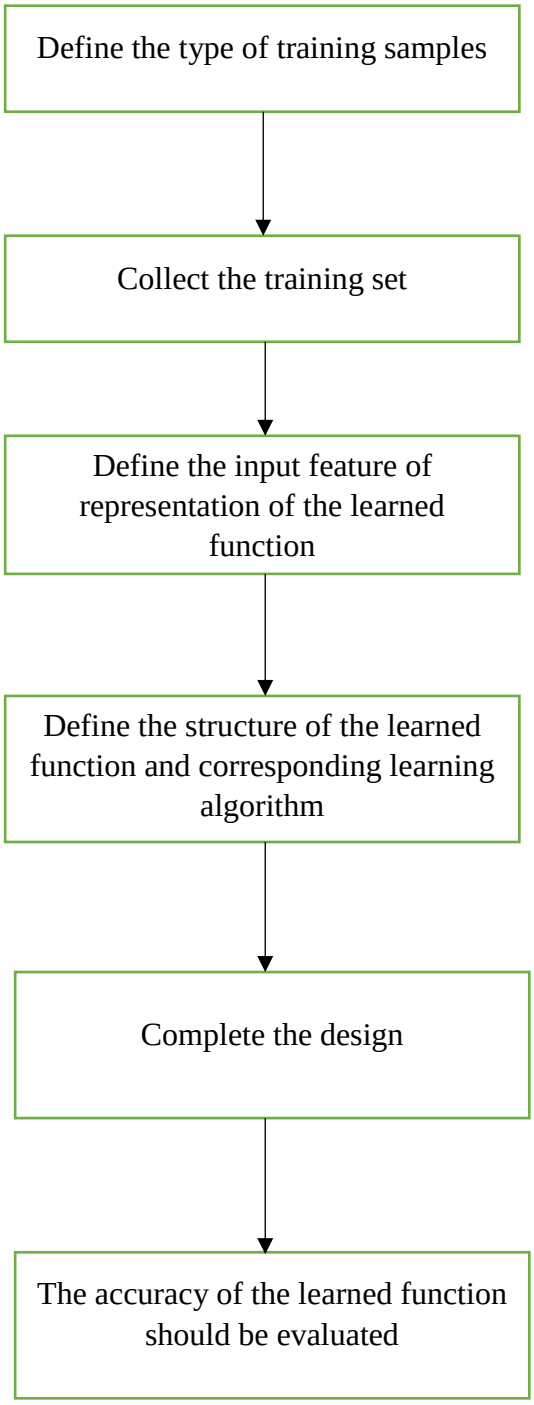


Figure 2.2. 1: Steps to solving supervised learning

2.2.2 Unsupervised learning

This form of classification is done without interpretive guidance from an analyst. The algorithm automatically organizes similar pixel values into groups that become the basis for different classes. This is entirely based on the statistics of the image data distribution and is often called clustering. The process is automatically optimized according to cluster statistics without the use of any knowledge-based control (i.e., ground referenced data). The method is, therefore, objective and entirely data-driven. It is particularly suited to images of targets or areas where there is no ground knowledge. Even for a well-mapped area, the unsupervised classification may reveal some spectral features which were not apparent beforehand. Figure 2.2.2 shows the necessary steps to solving unsupervised learning.

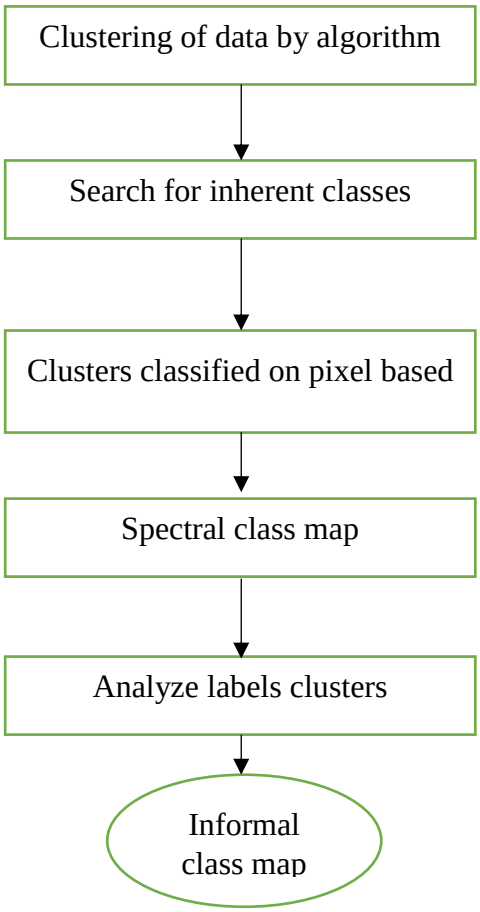


Figure 2.2. 2: Steps to solving supervised learning

2.3 Neural networks

A neural network is a system of interconnected artificial “neurons” that exchange messages between each other. The connections have numeric weights that are tuned during the training process so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting “neurons.” Each layer has many neurons that respond to different combinations of inputs from the previous layers. As shown in Figure 2.3a, the layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, and the third layer detects patterns of those patterns.

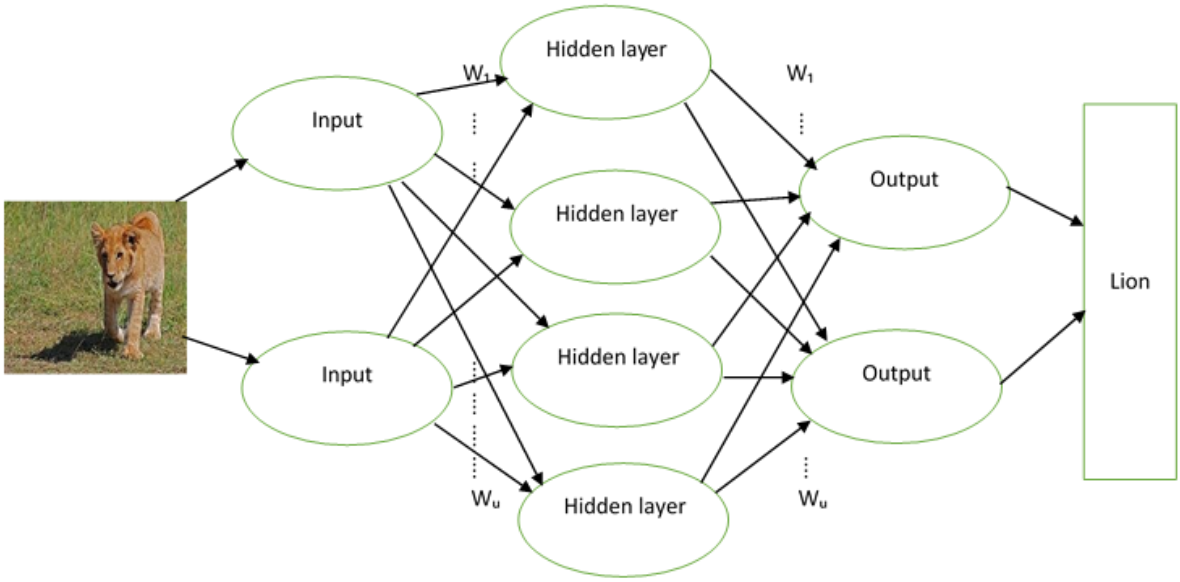


Figure 2.3 1: Feedforward neural network

The networks are inspired by biological neural systems whose basic computational unit is a neuron, and they are connected with synapses. Figure 2.3b compares a biological neuron with a basic mathematical model. They can also be applied to problems of prediction, classification or control in a broad spectrum of fields such as finance, cognitive psychology/neuroscience, medicine, engineering, and physics.

Neural networks are used when the exact nature of the relationship between inputs and output is not known. A key feature of neural networks is that they learn the relationship between inputs and output through training.

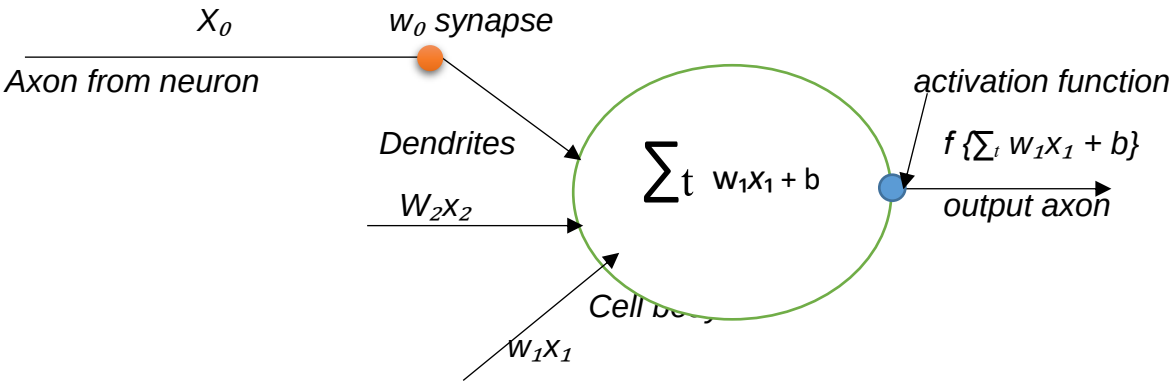


Figure 2.3 2: Mathematical model of a biological neuron

Some common examples of neural network training techniques are convolutional neural network (CNN), residual neural network (RESNET), etc. Some unsupervised network architectures are multilayer perceptron's, Kohonen networks and Hopfield networks, etc.

2.3.1 Convolutional Neural Network (CNN)

CNN is composed of one or more convolutional layers with fully connected layers (matching those in typical artificial neural networks) on top. It also uses tied weights and pooling layers. This architecture takes advantage of the 2D structure of input data. In comparison with other neural network architectures, convolutional neural networks have shown superior results in both image and speech, and pattern recognition applications. They can also be trained with a standard backpropagation learning algorithm. CNNs are easier to train than other regular deep Feed-forward neural

networks and have many fewer parameters to estimate, making them a highly attractive architecture to use.

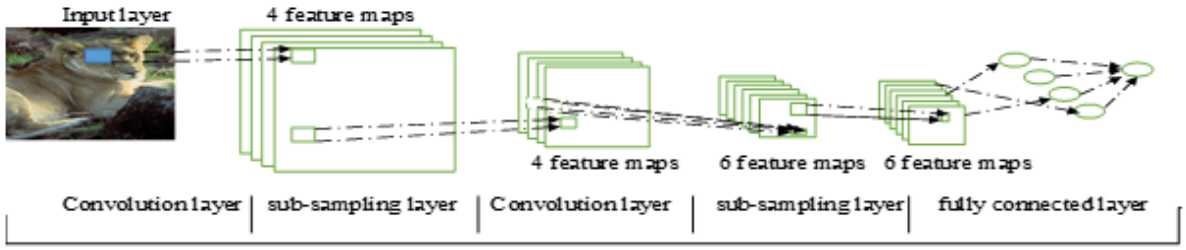


Figure 2.3. 1: Typical CNN

By stacking multiple and different layers in a CNN, complex architectures can be built for classification problems. Four types of layers are most common: convolution layers, pooling/subsampling layers, non-linear layers, and fully connected layers.

2.3.2 Multilayer perceptron (MLP)

The Multilayer Perceptron (MLP) consists of an input and an output layer with one or more hidden layers of nonlinearly-activating nodes or sigmoid nodes. It involves several compined perceptrons. The result is a network with several hidden layers between the input and the output ones, which can approximate nonlinear functions. In parallel to the introduction of more than one layer, the calculation of the feedforward values at each layer, and the weight adjustment method has been improved. Back Propagation algorithm is used to adjust the weights (Paliouras, 1993) similar to the one used in a simple perceptron, but incorporates more parameters. The aim is still to reduce the sum of least square errors for the training set, but the errors are now propagated more than one layers back to adjust all the weights and the thresholds in the network (Paliouras, 1993).

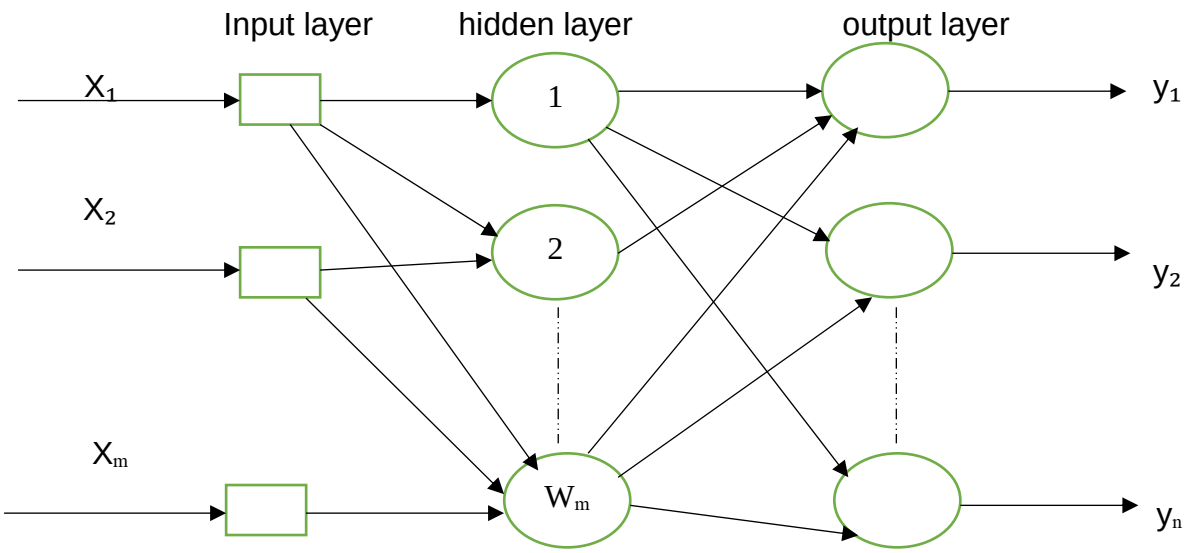


Figure 2.3. 2: Multilayer perceptron network

2.4 Review of similar works

(Koprinkova & Petrova, 1999) Presented a paper on data-scaling problems in feedforward neural-network training. In this paper, they pointed out that these problems appear when the experimental data to be learned to vary across a wide interval and has been scaled. One approach to solve this problem is to propose a parametric output function of the neurons as it will allow the introduction of new parameters into the network so that during the process of feedforward Propagation, parameters like the relative square error is minimized while the loss of information is almost avoided. However, this approach has some demerits as an increase in the network parameter exposes the network to overfitting which is undesirable in image classification as the network is designed to provide no room for appropriate scaling of its parameters.

(Zheng et al 2018) proposed a scalable deep CNN called S-Net in which the network scale can be adjusted dynamically in multitasking for real-time operations with minimal or negligible loss in performance. This approach offers a direct technique to assess performance gains with the network depth increased. However, the computational cost and long training procedures of the network is challenging and increasingly becoming unaffordable. More so, the chances of scaling some parameters in the network are slim as it's already designed for complex systems.

(Trnovszky et al. 2017) designed an Animal Recognition System Based on Convolutional Neural Network. The model was trained using a created animal dataset of five different classes with all animal images aligned and normalized based on the positions of the animal's eyes. Experimental results from the training and testing were used to evaluate the performance of the network, and then comparison was made with other image processing algorithms to access the effectiveness of the proposed algorithm. However, the network is composed of large weights, and that can be a sign of an unstable network where small changes in the input can lead to significant changes at the output, and that can be a sign of more complex network that has overfitted the training data.

(Alex Krizhevsky et al. 2007) trained a large deep neural network to classify 1.2 million images in the Image Net LSVRC-2010 contest into the 1000 different classes. The training of the Model was done on a single GTX 580 GPU having only 3GB of memory which limits the maximum size of the networks that can be trained. The CNN proposed in the research achieves a top-5 error rate of 18.2% averaging the predictions of five similar CNNs as measures to curb overfitting was employed with optimization on the

forefront. However, there is no established test set from the dataset as this may affect the accuracy of the network inappreciably.

(Guignard & Weinberger, 2016) Authored Animal identification from remote camera images using Snapshot Serengeti dataset, which consists of 3.2 million images of over 50 Species taken more than 5 decades in the Serengeti ecosystem in sub-Saharan Africa. However, images with pictures of human beings in the dataset make prediction very poor and thus affects validation accuracy.

(Jacobs et al 2017) authored a paper titled “Towards Scalable Parallel Training of Deep Neural Networks” where they propose a new framework for parallelizing deep neural network training that maximizes the amount of data that is ingested by the training algorithm. The proposed framework, called Livermore Tournament Fast Batch Learning (LTFB) targets large-scale data problems. The LTFB approach creates a set of Deep Neural Network (DNN) models and trains each instance of these models independently and in parallel. This new approach maximizes computation and minimizes the amount of synchronization required in training deep neural network, a significant bottleneck in existing synchronous deep learning algorithms.

However, in this research, we aim to find suitable parameter values for the model, which will give the best optimum performance characteristics through scalability. We propose to adjust the network parameters by intuition and systematic experimentation. More so, we will optimize floating point’s values from the results obtained by changing them to fixing points to reduce memory complexity and yield faster processing in the network while targeting animal prediction using a convolutional neural network.

Chapter three

Design and Methodology

3.0 Introduction

This chapter introduces the methodology used with the algorithm adopted and the proposed framework made to achieve the objectives. The chosen algorithm has been selected out of a large list of available ones for several important reasons. The main one is the fact that it has been used by many machine learning researchers and has contributed immensely to the field of computer vision. As a result, they become popular among machine learning researchers. Another reason is that recognition using the algorithm is rugged to distortions, such as a change in shape due to the camera lens and different lighting conditions.

3.1 Concept of Classification Technique

A classification technique is a systematic approach to building classification models from an input dataset. Examples include decision trees classifiers, neural networks, support vector machines, and naïve Bayes classifier. Each technique employs a learning algorithm to identify a model that best fits the relationship between the attributes set and class label of input data. The model generated by a learning algorithm should both fit the input data well and correctly predicts the class label of records it has never seen before (Pang-Ning Tan et al.). The general approach for solving a classification problem requires firstly a training set consisting of records whose class labels are known to be provided. Secondly, a test set to whose records of class labels are not known to be applied to the classification model generated by the training set.

3.2 Design and requirement phase

There are two ways of solving AI (Artificial intelligence) relate problems namely; hardware and software methods but preference is always giving to software design aspect as it is relatively cheap and does not involve a lot of complexity with regards to its architecture and requirements. Training and learning operations take place in the software design stage. The hardware, like general purpose processors and FPGA serve as an implementation platform for even more complex architectures. The proposed model was implemented using python programming language, particularly in spyder with keras and tensorflow as backend. Tensorflow is an open source deep learning framework created by Google that gives developers coarse control over each neuron so that weights can be adjusted to achieve optimal performance. As the task is relatively not intensive because of the small number of images in the dataset, a GPU was skipped and a CPU core i5 with 2.60GHz processing speed and a dedicated graphics card at the high end was used that can train an average of 94 samples per second.

3.3 Deep learning network for recognition and identification

Deep convolutional neural networks (CNNs) are a specialized kind of ANNs that use convolution in place of general matrix multiplication in at least one of their layers. In contrast to simple neural networks that have one or several hidden layers, deep CNNs consist of many layers, as shown in figure 3.3. Such a feature allows them to represent highly nonlinear and varying functions compactly. CNNs involve many connections, and the architecture is typically comprised of different types of layers including convolution, pooling, fully connected layers, and a realize form of regularisation. To learn complicated features and functions that can represent high-level abstractions

(e.g., in vision, language, and other AI-level tasks), CNNs would need deep architectures. Deep architectures and CNNs consist of a large number of neurons and multiple levels of latent calculations of non-linearity. Each level of architecture of CNN represents features at a different level of abstraction defined as a composition of lower level features (Namatēvs, 2018).

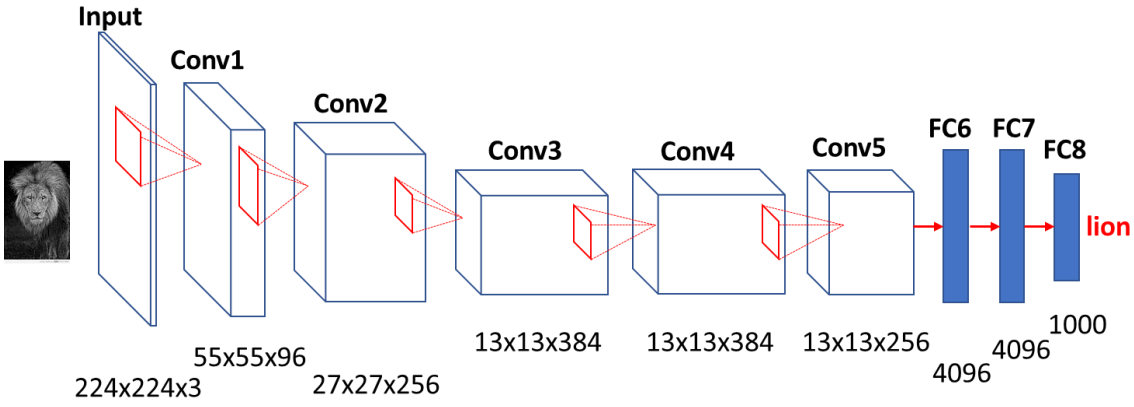


Figure 3. 1: Deep CNN Architecture

3.4 Proposed neural network model

After identifying some shortcomings (network complexity, number of parameters, computational cost) in quite a handful number of related works reviewed, we decided to propose and implement a network capable of overshadowing the drawbacks mentioned above. The proposed network is an adopted CNN based architecture with lightweight edge parameters for animal identification and recognition, as shown in figure 3.4. The choice of the number of layers for the network is for the benefit of more nonlinearity which will be added to the network technically it implies that the network gets more and more powerful to learn complex data when given samples. The network designed is scalable such that the performance does not deteriorate even though the system gets large.

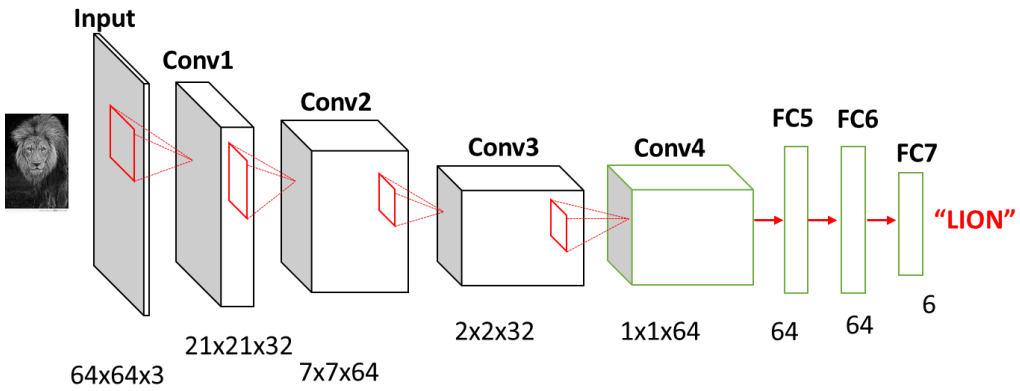


Figure 3. 2: Proposed network design

3.5 Building blocks of Deep CNN

The deep CNN architecture includes several building blocks such as convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers, a pooling layer and followed by a fully connected layer (Yamashita et al. 2018).

3.5.1 Convolution layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function (Yamashita et al., 2018). Convolution is a specialized type of linear operation used for feature extraction where a small array of numbers called a kernel is applied across the input which is an array of numbers called a tensor. An element-wise product between each element of the kernel as shown in figure 3.5.1 and the input tensor is calculated at each location of

the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map (Yamashita et al., 2018). This

Procedure is repeated applying multiple kernels to form an arbitrary number of feature maps which represent different characteristics of the input tensors.

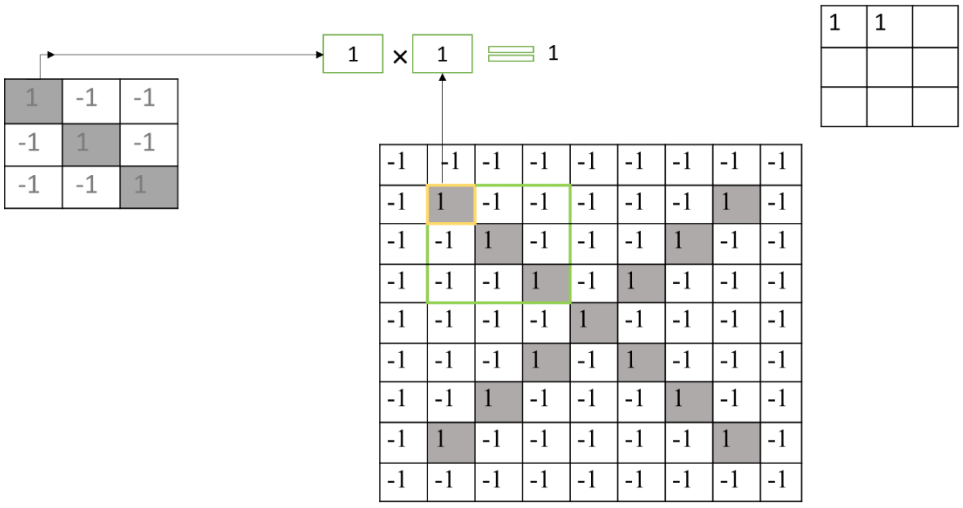


Figure 3.5. 1: Convolution layer operation

3.5.2 Pooling layer

A pooling layer provides a typical downsampling operation which reduces the in-plane dimensionality of the feature maps to introduce a translation invariance to small shifts and distortions and decrease the number of subsequent learnable parameters (Yamashita et al., 2018). Max pooling is the most popular form of pooling operation, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values as shown in figure 3.5.2. A max pooling with a filter of size 2 × 2, with a stride of 2 is commonly used in practice (Yamashita et al., 2018).

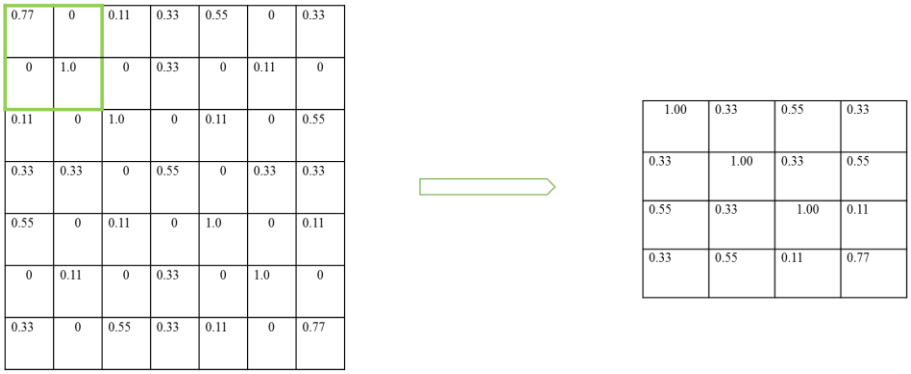
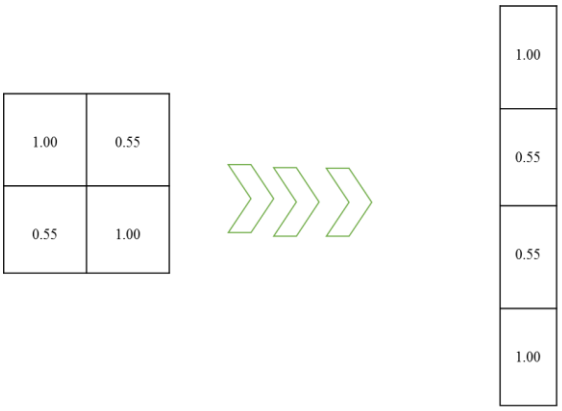


Figure 3.5. 2: Max pooling operations with 2 by 2 filter stride

3.5.3 Fully connected layer

This is where the actual classification is done. The output feature maps of the final convolution or pooling layer is typically flattened as shown in figure 3.5.3 i.e., transformed into a one-dimensional (1D) array of numbers (or vector) and connected to one or more fully connected layers also known as dense layers in which every input is connected to every output by a learnable weight (Yamashita et al., 2018). The final fully connected layer typically has the same number of output nodes as the number of classes (Yamashita et al., 2018).



30

Figure 3.5. 3: Fully connected layer 1D dimensional array

3.5.4 Activation functions

Activation functions in the hidden layer help in mapping the non-linearity relationship between input and output. There are several activation functions as shown in figure 3.5.4 however, commonly used activation functions in hidden layers are sigmoid and Relu. There is no rule for applying specific activation functions. Different activation functions need to be evaluated for specific datasets. In this research, we use the rectified linear (also referred to as 'Relu') activation function because it facilitates (Singaravel et al 2018) model training using gradient-based optimization methods.

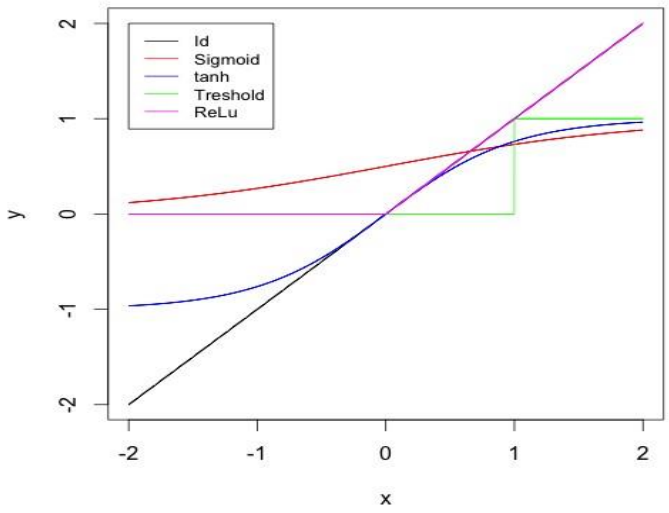


Figure 3.5. 4: Several activation functions graph

3.5.5 Last layer activation function

The activation function applied to the last fully connected layer is usually different from the others. An appropriate activation function is selected according to each task. An activation function applied to the multiclass classification task is a softmax function as used in this research which normalizes output real values from the last fully connected

layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1 (Yamashita et

al., 2018). Typical choices of the last layer activation function for various types of tasks are summarized in Table 3.5.5

Table 3.5. 5: A list of commonly applied last layer activation functions for various tasks

Task	Last layer activation function
Multiclass single-class classification	Softmax
Multiclass classification	Sigmoid
Regression to continuous values	Identity

3.6 Training deep CNN

Training deep architectures is a challenging task, and traditional methods that have proved useful when applied to uncomplicated neural network architectures are not as effective when applied to deep architecture. The training function means to use an overall algorithm that is used to train a neural network to recognize a specific input and map it to a specific output (Namatēvs, 2018). The most costly part of deep neural networks training is knowing the features and accessibility to labeled data.

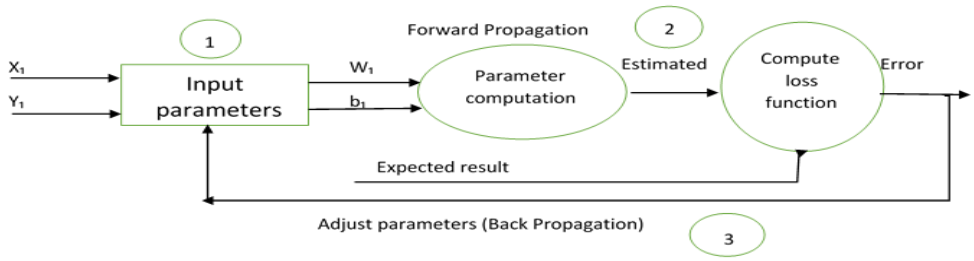


Figure 3. 6: Training process of a neural network

Learning process in deep neural networks involves calculating the gradients of complex functions and decides how they will be manipulated. CNNs are usually trained by back propagation algorithm (BP) and Stochastic Gradient Descent (SGD) to find weights and biases that minimize specific loss function as shown in figure 3.6 to map the random inputs to the targeted outputs as closely as possible.

3.6.1 Loss function

A loss function also referred to as a cost function, or error function measures the difference between output predictions of the network through forwarding propagation and given the expected result. Most commonly used loss function for multiclass classification is cross entropy, whereas mean squared error is typically applied to regression related problems.

3.6.2 Gradient descent

Gradient descent is commonly used as an optimization algorithm that iteratively updates the learnable parameters, i.e., kernels and weights of the network so as to minimize the loss. The gradient of the loss function provides direction in which the function has the steepest rate of increase and each learnable parameter is updated in the negative direction of the gradient with an arbitrary step size determined based on a hyperparameter called learning rate. The gradient is mathematically a partial derivative of the loss with respect to each learnable parameter and a single update of a parameter is formulated as follows:

$$W := w - \alpha * \frac{\partial L}{\partial w}$$

Where *w* stands for each learnable parameter, *α* stands for a learning rate, and *L* stands for a loss function (Yamashita et al., 2018).

3.7 Dataset description

Dataset was created using 6 images of different classes with each class having a number of images of more than 200 amounting to 2000 sets of images in total even though images in the dataset set have large variations in scale, pose and lighting. These images are in RGB and have undergone pre-processing to ensure a uniform input of 64 ×64 is fed into the neural network. For this research, only 760 images were used as a result of lack of good GPU with the required memory and processor speed to train the model with all the collected dataset. The new dataset created is to train the model to be more generic and have some good degree of universality even with other already made and extracted images from dataset.

3.8 System architecture of deep CNN for wildlife recognition

Deep learning has outperformed other machine learning algorithms in the area of image classification (Singaravel et al., 2018). The architecture involves mainly three layers, namely; input layer, hidden layer, and the output layer with the number of hidden layers defining the depth of the architecture. As shown in figure 3.8, the data pre-processing serves as the input layer; the hidden layers are the processes taking place in the deep learning and training block. However, optimization may be involved in the results from the hidden layers for correct prediction at the output layer.

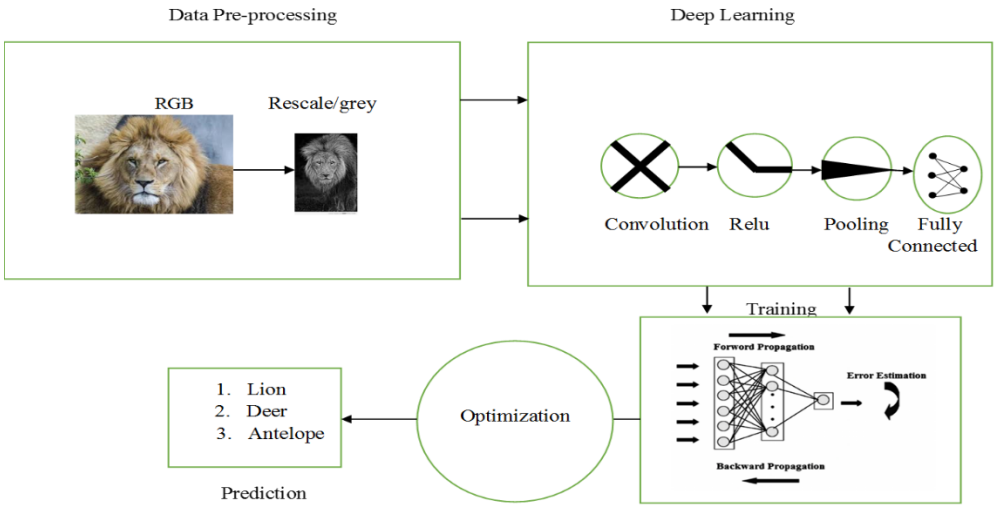


Figure 3. 8: Deep CNN architecture for wild life animal recognition

3.9 Implementation details

We took 2150 images for six classes both from camera snapshot in some few wildlife parks within Abuja, Nigeria and complimented it with some gotten using Google search engine for different classes as required. Many photos were of animals far in the distance obstructed by objects, or only partly in the frame, we manually screened these out from our training set even though it proved to be time-consuming so that images will not be misclassified. Instead of the six animal types in the complete dataset, we chose to first train on three classes: Wild pig, Rhino and Bear then took 344 images of rhino, 214 images of wild pig and 212 of bear from the dataset. Finally, images were cropped and resized to the same dimensions.

3.9.1 Parameters of the network

We implemented the network using machine learning package tensorflow and keras, which is built on top of theanos. The input layer to the network was the raw image from camera Snapshots, which is in RGB form with 20% dropout been applied to the layer. The network hidden layers for the first model designed consisted of two fully connected layers, and the second model three fully connected layers with a 50% drop out after each layer and Relu activation function to model non-linearities. The number of nodes in each fully connected layer was set to roughly three times the number of pixels in the image. The output layer consisted of three nodes, one for each classification and applied the softmax function to simulate a probability for each class. The error was measured using cross entropy loss and stochastic gradient descent, a learning rate of 0.01, and the momentum of 0.9 was used for optimization. The network was run for ten epochs on input image sizes 64 x 64 pixels although memory constraints limited the input image sizes to a maximum of 150 x 150 pixels for training. The network was run on an Intel i5 2.60 GHz processor with 8GB of memory.

Chapter four

Results and discussions

4.0 Introduction

This chapter presents the results of our proposed design and implementation. Firstly, we examine the overall model accuracy, made comparisons between different architectural design and between different types of images. Secondly, we present the loss and accuracy graph generated together with a summary of the network designed the number of parameters, including the computational time during the training process. Finally, we make a conclusion based on the outcome of our different design models.

4.1 Results for 4 convolutions with two output layers

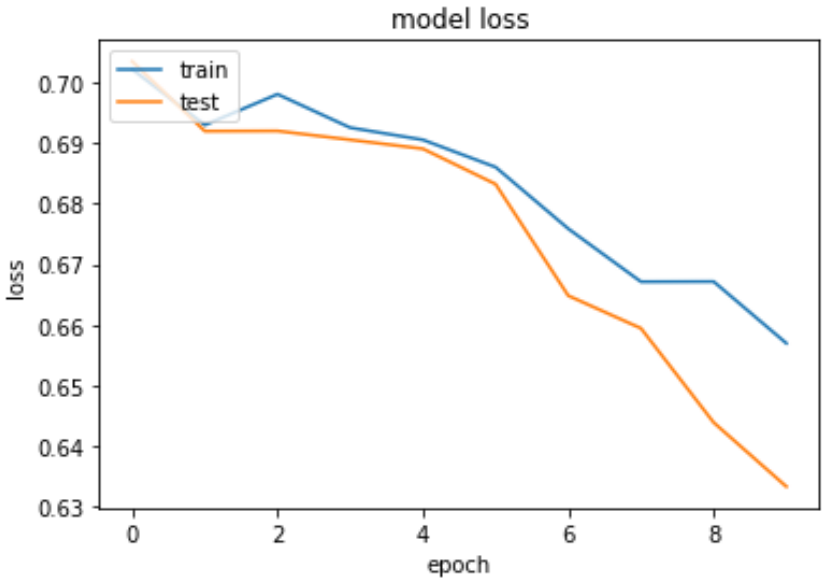


Figure 4.1 1: Loss graph

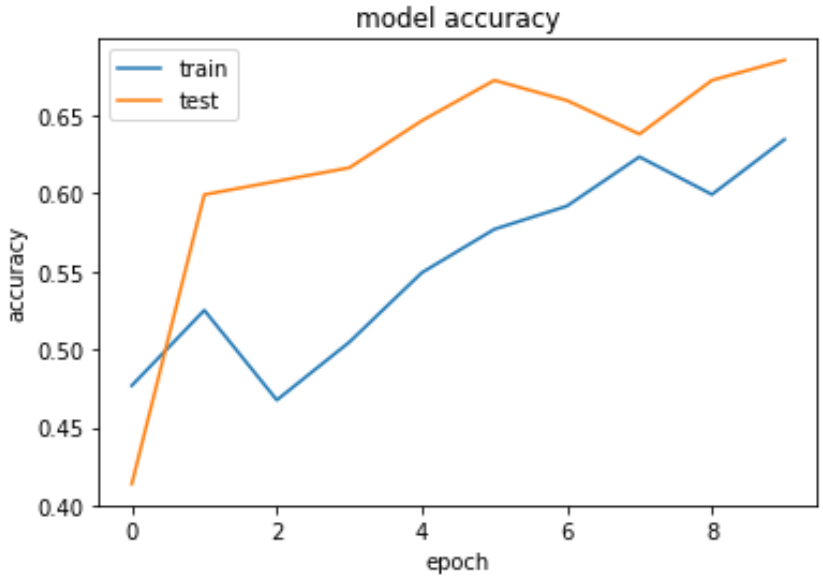


Figure 4.1 2: Accuracy graph

4.2 Results for 3 convolution with 3 output layers

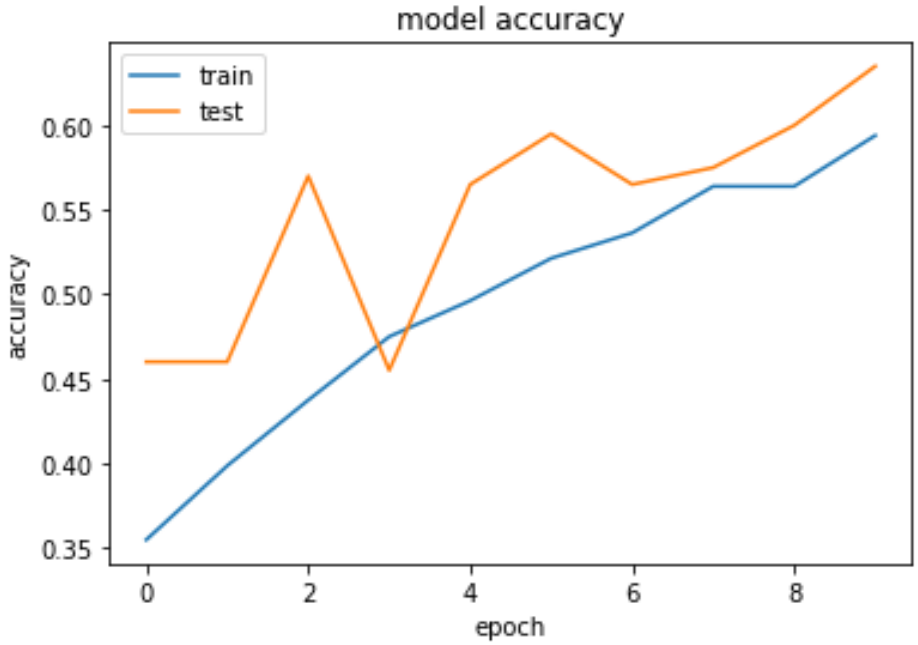


Figure 4.2 1: Accuracy graph

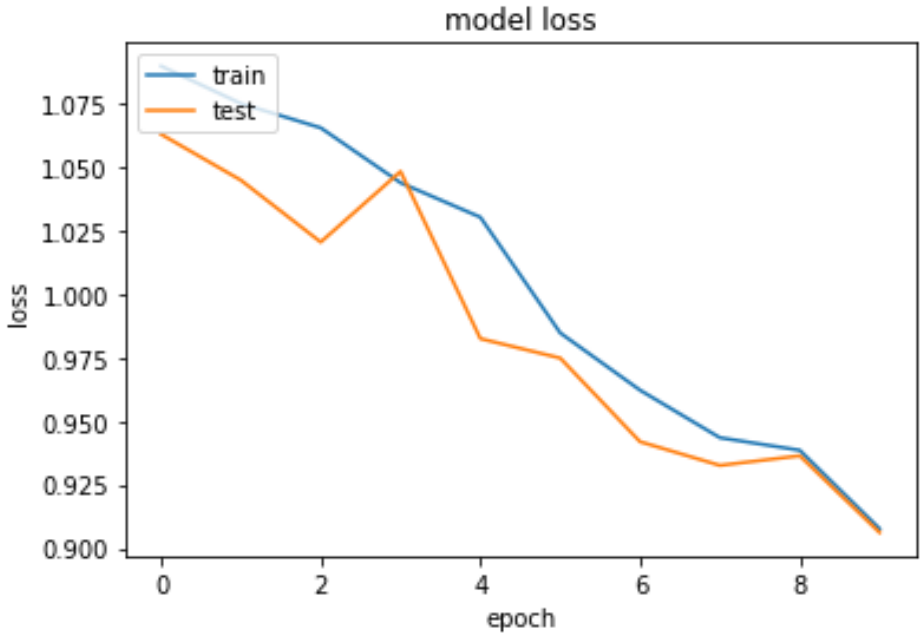


Figure 4.2 2: Loss graph

4.3 Results summary

Table 4. 3: Results summary

Number of layers	Validation Accuracy (%)	Validation Loss (%)	Training Loss	Execution time (s)	Training Accuracy (%)	Number of parameters
4convet, 2FC(128)	68.97	0.6161	0.6525	18	61.78	1,688,098
3Convet, 3FC(512)	63.50	0.9060	0.9077	29	59.40	7,744,643

4.4 Results discussion

In this work, we measure the performance (accuracy, loss, and computation time) of different modeled networks. We used three classes of images from the datasets and also hand-picked randomly 24 images from the dataset for testing after training the network. We report the accuracy and loss obtained by plotting their graph against the number of times the network is trained with the computation time in all cases. A detailed summary of the results is given in table 4.3. It can be seen that the two different architecture designed and implemented achieved relatively a good accuracy. But, the architecture with four convolution layers and two output layers shown in Figure 4.1 3, achieves better performance with regards to accuracy of 68% as compared to 64% in figure 4.2.1 after ten epochs and has the number of parameters in the network very much less as compared to the second network architecture.

The losses in the two different network architecture is dropping steadily. However, the first network architecture has a better loss percentage of 0.62 as shown in figure 4.1.1 which implies that the network design has less overfitting (memorizing) and is nowhere going to overfit if same parameters and datasets is maintained as compared to the second network architecture with 0.93 percentage loss after 10 epochs.

Chapter five

Conclusion and future works

5.0 Introduction

This chapter summarises the results presented in the thesis and concludes their importance in the context of recognition and identification.

5.1 Conclusions

In this research, we proposed a neural network software architecture and performed its evaluation. We employed the use of deep learning techniques to identify wildlife animals while ensuring that a robust system that can generalize our images (from the datasets) is realized. A manually created dataset from raw images fetched from the camera(s), preprocessed using some python libraries was used as inputs to the designed network. Many network architecture design with scalability was performed, but the two most important ones that showed good results was picked for evaluation.

During the evaluation, the proposed architecture with 4 convets and 2 output layer achieved good results while predicting or recognizing wildlife animals. Our approach could be used in both remote and urban areas to help prevent or reduce the number of the animal-vehicle collision, animal-humans attack, and animal crop destruction by detecting the presence of animal so that warning may be issued with a view of safety purpose. This research though, could consolidate other findings made in the recognition of wildlife animals and thus will help show the adaptability of deep CNN to even small datasets in their raw form captured from the camera(s). It will, in turn, lead to the formations of standard design philosophies that will make image recognition algorithms more practicable to solving real-life problems.

5.2 Future work

There are several areas this research has not been able to cover due to lack of time chief among other drawbacks and resources (e.g., lack of large datasets, High GPU Processor). However, the following issues are more specifically of interest.

1. The memory complexity optimization which could be done by changing the floating point values from the results obtained to fixing points. The goal is to reduce memory complexity and yield faster processing in the network while targeting animal prediction using a convolutional neural network.
2. Optimizing Execution time (a function of the number of MAC operations) by reducing the number of operations taking place in the MAC (multiply and accumulate) convolution layer. More so this will also reduce the computation complexity of the network and hence even the power consumption.

3. Accuracy of the network is a function of the amount of dataset fed, and this can be improved and maintained by collecting a large number of datasets which are preprocessed properly or discretized as this will enhance the accuracy of both training and validation.

4. Further making the network more scalable will make it achieve a good balance between latency, precision (exactness towards prediction) and hardware complexity as these are factors that define how efficient a neural network architecture can perform in hardware(s)

REFERENCES

Alex Krizhevsky, Ilya Sutskever, G. E. H. (2007). ImageNet Classification with Deep Convolutional Neural Networks. *Handbook of Approximation Algorithms and Metaheuristics*, 60-1-60–16. https://doi.org/10.1201/9781420010749

Anupam Anand. (2018). *Unit 13 Image Classification*. (May), 41–58.

Guignard, L., & Weinberger, N. (2016). *Animal identification from remote camera images*. 1–4.

Jacobs, S. A., Dryden, N., Pearce, R., & Van Essen, B. (2017). *Towards Scalable Parallel Training of Deep Neural Networks*. (Sc 17), 1–9. https://doi.org/10.1145/3146347.3146353

Koprinkova, P., & Petrova, M. (1999). Data-scaling problems in neural-network training. *Engineering Applications of Artificial Intelligence*, *12*(3), 281–296. https://doi.org/10.1016/S0952-1976(99)00008-1

Learning, C. (2017). *) Machine Learning (* □□□□□□ □□□□□□□□.

Namatēvs, I. (2018). Deep Convolutional Neural Networks: Structure, Feature Extraction and Training. *Information Technology and Management Science*, *20*(1), 40–47. https://doi.org/10.1515/itms-2017-0007

Paliouras, G. (1993). Scalability Of Machine Learning Algorithms. *Neural Networks*, (November).

Pang-Ning Tan et Al. (n.d.). <*Classification Decision Trees Etc.Pdf*>.

Shrivastav, U., & Singh, S. K. (2019). *Digital Image Classification Techniques*. (October), 162–187. https://doi.org/10.4018/978-1-5225-9096-5.ch009

Singaravel, S., Suykens, J., & Geyer, P. (2018). Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction. *Advanced Engineering Informatics*, *38*(June), 81–90. https://doi.org/10.1016/j.aei.2018.06.004

Trnovszky, T., Kamencay, P., Orjesek, R., Benco, M., & Sykora, P. (2017). Animal recognition system based on convolutional neural network. *Advances in Electrical and Electronic Engineering*, *15*(3), 517–525. https://doi.org/10.15598/aeee.v15i3.2202

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9

Zheng, B., Sun, R., & Tian, X. (2018). S-Net: a scalable convolutional neural network for JPEG compression artifact reduction. *Journal of Electronic Imaging*, *27*(04), 1. https://doi.org/10.1117/1.jei.27.4.043037

黄正华. (2014). 矩阵分析. *447*(1992), 51–55.

Murakami, R., Okuyama, Y., & Abdallah, A. Ben. (2018). *"Animal Recognition and Identification with Deep Convolutional Neural Networks for Farm Monitoring", Information Processing Society Tohoku Branch Conference, Koriyama, Japan.*

The H. Vu, Okuyama, Y., & Abdallah, A. Ben. (2019), *''Analytical performance assessment and high-throughput low-latency spike routing algorithm for spiking neural network systems," The Journal of Supercomputing. DOI: https://doi.org/10.1007/s11227-019-02792-y*

The H. Vu, Abdallah, A. Ben. (2019), *"A Low-latency K-means based Multicast Routing Algorithm and Architecture for Three Dimensional Spiking*

Neuromorphic Chips", IEEE International Conference on Big Data and Smart Computing (BigComp 2019), Kyoto, Japan, Feb 28 - Mar 2, 2019 [best paper award]

The H. Vu, Murakami, R., Okuyama, Y., & Abdallah, A. Ben. (2018). *"Efficient Optimization and Hardware Acceleration of CNNs towards the Design of a Scalable Neuro-inspired Architecture in Hardware", IEEE International Conference on Big Data and Smart Computing (BigComp 2018), Shanghai, China, January 15-18, 2018.*

Murakami, Y., Okuyama, Y., & Abdallah, A. Ben. (2018). *SRAM Based Neural Network System for Traffic-Light Recognition in Autonomous Vehicles Paper Contribution System Architecture Preliminary Evaluation Conclusion and Future work*. 1–27.

APPENDIX 1

CODES FOR 3 CONVETS, 3 FULLY CONNECTED LAYER

```
import numpy
import matplotlib.pyplot as plt
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import TensorBoard


NAME = ' wild life animals classification-cnn-64x2-{}'.format(int(time.time()))
tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))

K.set_image_dim_ordering('tf')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

def pre_process(X):

    # normalize inputs from 0-255 to 0.0-1.0
    X=X.astype('float32')
    X = X / 255.0
    return X

def one_hot_encode(y):

    # one hot encode outputs
    y = np_utils.to_categorical(y)
    num_classes = y.shape[1]
    return y,num_classes

def define_model(num_classes,epochs):
    # Create the model
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), padding='same',
activation='relu', kernel_constraint=maxnorm(3)))
    '''model.add(Conv2D(64, (3, 3), activation='relu'))'''
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
```

```
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_constraint=maxnorm(3)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Dense(64, activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    lrate = 0.01
    decay = lrate/epochs
    sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
    model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    print(model.summary())
    return model


# load data
X,y=load_data.load_datasets()

# pre process
X=pre_process(X)

#one hot encode
y,num_classes=one_hot_encode(y)


#split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=7)

epochs = 10
#define model
model=define_model(num_classes,epochs)


# Fit the model
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs,
batch_size=32, callbacks=[tensorboard])

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# serialize model to JSONx
model_json = model.to_json()
with open("model_face.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_face.h5")
print("Saved model to disk")
```

APPENDIX 2

CODES FOR 4 CONVETS, 2 FULLY CONNECTED LAYER

```
import numpy
import matplotlib.pyplot as plt
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
import load_data
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.callbacks import TensorBoard


NAME = ' wild life animals classification-cnn-64x2-{}'.format(int(time.time()))
tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))

K.set_image_dim_ordering('tf')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

def pre_process(X):

    # normalize inputs from 0-255 to 0.0-1.0
    X=X.astype('float32')
    X = X / 255.0
    return X

def one_hot_encode(y):

    # one hot encode outputs
    y = np_utils.to_categorical(y)
    num_classes = y.shape[1]
    return y,num_classes

def define_model(num_classes,epochs):
    # Create the model
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), padding='same',
activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_constraint=maxnorm(3)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Dense(128, activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    lrate = 0.01
    decay = lrate/epochs
    sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
    model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    print(model.summary())
    return model


# load data
X,y=load_data.load_datasets()

# pre process
X=pre_process(X)

#one hot encode
y,num_classes=one_hot_encode(y)


#split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=7)

epochs = 10
#define model
model=define_model(num_classes,epochs)


# Fit the model
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs,
batch_size=32, callbacks=[tensorboard])

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# serialize model to JSONx
model_json = model.to_json()
with open("model_face.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_face.h5")
print("Saved model to disk")
```