A Project/Dissertation Report

on

Bird Watcher Using Deep Learning Platform

Submitted in partial fulfillment of

the requirement for the award ofthe

degree of

B.Tech (CSE)

Under The
Supervision of
Dr. A John

Submitted By

Udit Raj Singh (18SCSE1010299)

Anil Bameta (18SCSE1010185)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING GALGOTIAS UNIVERSITY, GREATER NOIDA,
INDIA

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Birdwatching is a common hobby but to identify their species requires the assistance of bird books. To provide birdwatchers a handy tool to admire the beauty of birds, we developed a deep learning platform to assist users in recognizing 27 species of birds endemic to Taiwan using a mobile app named the Internet of Birds (IoB). Bird images were learned by a convolutional neural network (CNN) to localize prominent features in the images. First, we established and generated a bounded region of interest to refine the shapes and colors of the object granularities and subsequently balanced the distribution of bird species. Then, a skip connection method was used to linearly combine the outputs of the previous and current layers to improve feature extraction. Finally, we applied the softmax function to obtain a probability distribution of bird features. The learned parameters of bird features were used to identify pictures uploaded by mobile users. The proposed CNN model with skip connections achieved higher accuracy of 99.00 % compared with the 93.98% from a CNN and 89.00% from the SVM for the training images. As for the test dataset, the average sensitivity, specificity, and accuracy were 93.79%, 96.11%, and 95.37%, respectively.

\

INTRODUCTION

1.1 BACKGROUND

The everyday pace of life tends to be fast and frantic and involves extramural activities. Birdwatching is a recreational activity that can provide relaxation in daily life and promote resilience to face daily challenges. It can also offer health benefits and happiness derived from enjoying nature [2].

Numerous people visit bird sanctuaries to glance at the various bird species or to praise their elegant and beautiful feathers while barely recognizing the differences between bird species and their features. Understanding such differences between species can enhance our knowledge of exotic birds as well as their ecosystems and biodiversity [1].

However, because of observer constraints such as location, distance, and equipment, identifying birds with the naked eye is based on basic characteristic features, and appropriate classification based on distinct features is often seen as tedious [1]. In the past, computer vision, and its subcategory of recognition, which use techniques such as machine learning, have been extensively researched to delineate the specific features of Objects, including vegetables and fruits, landmarks, clothing, cars, plants, and birds, within a particular cluster of scenes.

However, considerable room for improvement remains in the accuracy and feasibility of bird feature extraction techniques [2]. Detection of object parts is challenging because of complex variations or similar sub ordinate categories and fringes of objects [2]. Intraclass and inter- class variation in the silhouettes and appearances of birds is difficult to identify correctly because certain features are shared among species.

1.2 PROBLEM STATEMENT

Various kind of bird species are getting on the verge of extinction. Because of continuous deforestation, global warming, air pollution and many human activities, we need to have continuous monitoring of some categories of bird. With the use of proposed architecture we can make prediction of birds and these images can be given from users [2].

1.3 OBJECTIVE

- To classify the aesthetics of birds in their natural habitats, this studydeveloped a method using a convolutional neural network (CNN) to extract information from bird images captured previously or in real time by identifying local features.
- First, raw input data of myriad semantic parts of a bird were gathered and localized. Second, the feature vectors of each generic part were detected and filtered based on shape, size, and color [3].
- Third, a CNN model was trained with the bird pictures in a graphics processing unit (GPU) for feature vector extraction with consideration of the afore- mentioned characteristics, and subsequently the classified, trained data were stored on a server to identify a target object [4].
- Ultimately, information obtained from a bird image uploaded by an end-user, captured using a mobile camera, can be navigated through the client–server architecture to retrieve information and predict bird species from the trained model stored on the server. This process facilitates efficient correlation of fine-grained object parts and autonomous Bird identification from captured images and can contribute considerable, valuable information regarding bird species [3].

LITERATURE SURVEY

2.1 SUMMARY OF PRIOR WORKS

- Recently, some fine-grained visual categorizations methods have been proposed for species identification, and they have become a promising approach within computer vision research, with applications in numerous domains [5].
- Numerous fine-grained recognition datasets, such as ImageNet, ILSVRC, Caltech-256, and CUB 200, have trained models with a wide variety of data to extract global featuressuch as colors, textures, and shapes from multilabel objects [5].
- Many approaches have been applied for generic object recognition. Some methods applylocal part learning that uses deformable part models and region-CNN for object detection, generation of a bounding box, and selection of distinctive parts for image recognition [6].
- Some studies have focused on discriminative features based on the local traits of birds. Simultaneous detection and segmentation are used to localize score detections effectively. Pose-normalization and model ensembles are also used to improve the performance of fine-grained detection by generating millions of key point pairs through fully convolutional search.
- Discriminative image patches and randomization techniques are integrated to distinguish classes of images and prevent overfitting. The present work also approached the learning of discriminative image features using a CNN architecture for fine-grained Recognition [7]. However, a complementary approach using domain knowledge of general bird features was integrated to provide detailed information about the predicted bird [7].
- The advancement of consumer products, such as smart- phones, digital cameras, and wearable gadgets, has transformed multidisciplinary approaches toward technology by connecting the physical and digital worlds [8].
- High-resolution Digital cameras in smartphones are the most pervasive tools used for recognizing the salient features of physical objects, enabling users to detect, identify objects and share related knowledge. Birds present in a flock are often deeply colorful .

- Therefore, identification at a glance is challenging for both birdwatchers and onlookers Because of birds' ambiguous semantic features [9]. To address this problem, an informationretrieval model for Birdwatching has been proposed that uses deep neural networks to localize and clearly describe bird features with the aid of an Android smart- phone [10].

2.2 OUTCOME OF THE REVIEW

- ❖ Automatic classification of bird species from bird image samples has recently attracted the interest of the research community because of the improvement of different techniques in this field [11].
- ❖ Signal processing and Machine learning
- ➢ Signal processing is a broad term that involves the use of image processing techniques to improve signal quality and extract a set of features from the image signal.
- ➢ Machine learning algorithms use these features to develop decision methods that can predict and classify the image patterns.

SIGNAL PROCESSING

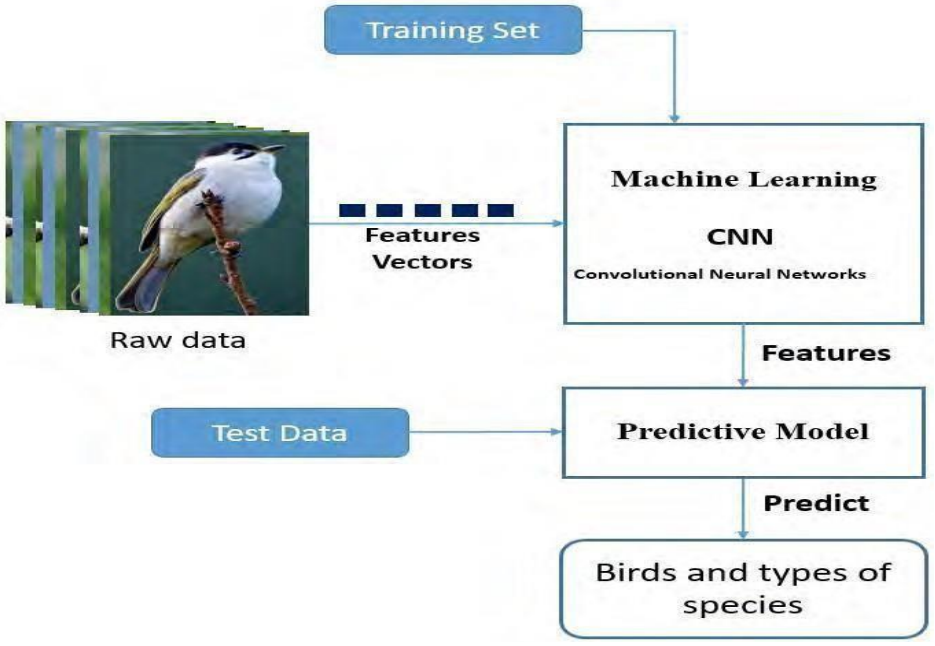


Figure 1.1 – Signal processing for bird images

2.3 DATA ACCUSTION

Feature extraction is vital to the classification of relevant information and the differentiation of bird species. We combined bird data fromthe Internet of Birds (IoB) and an Internet bird dataset to learn the bird species [1].

IOB

The IoB is a crowd sourced met search-engine database specifically for birds, where any individual can store bird images and instantly retrieve information about the birds therein. Uploaded bird images are identified from extracted features. This platform encourages individuals to become involved in Birdwatching and to enrich their knowledge of various bird species [1].

The IoB is available online for free (with keyword: Who Cares? Keep Walking). Fig. 1 shows the app interface. Because a fall detection module is embedded in the system, the app also serves as a wellness platform to assist individuals in staying safe while Birdwatching [13]. In addition, the system can track the distance individuals cover from their daily physical strides using a pedometer to promote fitness and motivate users to walk while Birdwatching [13].

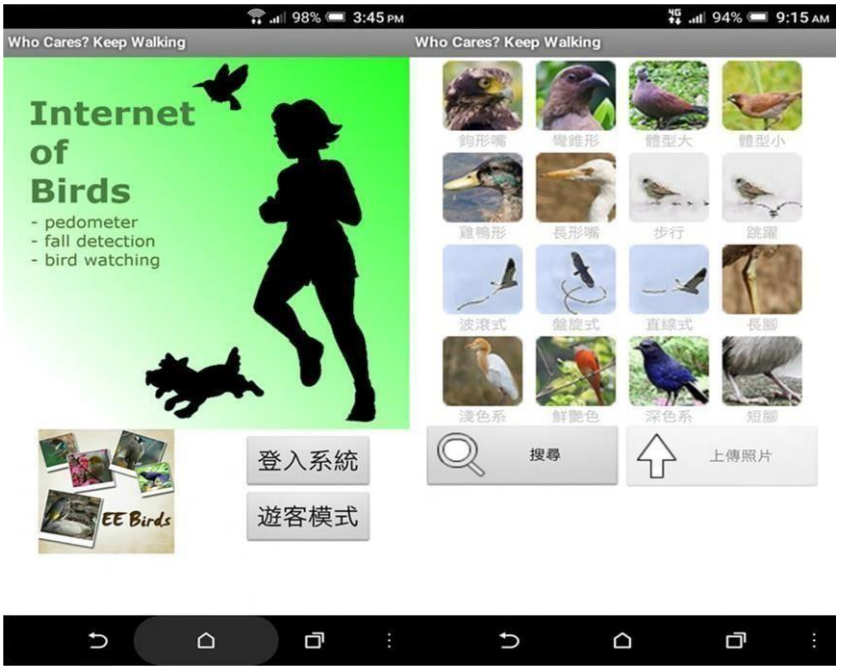


Figure 1.2 – IOB interface

INTERNET BIRD IMAGES

A pool of images is required for deep learning of sub categorization. Bird images containing27 bird species endemic to Taiwan on various backgrounds were compiled from the IoB andseveral other online resources. The use of public-domain images has benefits and drawbacks [13].

Although Internet Image sources add diversity to the dataset, the images may be contaminated with noise, harshness, spurious pixels, and blurred parts, all of which degrade image quality. Therefore, to limit the intensity of deformity in an assortment of images, high-pixel images with clear boundaries were used [14]. Finally, to obtain standardized balancein the dataset, the bird species images were transformed and augmented as follows:

- Random flipping: Images were horizontally and vertically flipped.
- Rotation: Images were randomly rotated (maximum angle of 25◦) for training.
- Translation: Images were randomly shifted -10 to 10pixels.
- Zero-phase component analysis whitening: Dimension and redundancy in the matrix of pixel images were decreased.
- Gaussian filtering: Images were blurred for effective smoothing of noise.

In deep learning algorithms, feature extraction is a generalization step to differentiate the learning categories of input data patterns.

Object recognition with high-level feature extraction architecture comprises the following steps:

(1) data content analysis, in which all generic raw data are pre-processed to extract nonlinear transformations and to fit the parameters into a machine learning model for feature extraction.

(2) Optimal probabilities of relevant structural information from each tuned parameter are clubbed into a new array ofclassifiers.

(3) A prediction is made based on trained and learned parameters.

To extract multiple feature levels from raw data and evaluate the performance of the CNN for the dataset, the dataset was split into the three modules discussed as follows:

(1) The training dataset comprised raw data samples that were incorporated into the training model to determine specific feature parameters, perform correlational tasks, and create a related classification model.

(2) The validation dataset was used to tune hyper parameters of the trained model to minimize over fitting and validate performance [13]. The model regularizes early stopping to prevent over fitting and to enhance learning when the precision of the training dataset increases while the error ofthe validation dataset remains the same or decreases [14].

(3) The test dataset was used to test the classifier parameters and assess the performance of the actual prediction of the network model [15]. Once the features had been extracted from the raw data, the trained prediction model was deployed to classify new input images. Fig. 1 shows the module for extracting unique features of birds with the CNN and predicting the most classified labels for the input images [16].

NUMPY

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

WHY USE NUMPY?

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

PANDAS

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself.[4] Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

Developer Wes McKinney started working on pandas in 2008 while at AQR Capital Management out of the need for a high performance, flexible tool to perform quantitative analysis on financial data. Before leaving AQR he was able to convince management to allow him to open source the library.

Another AQR employee, Chang She, joined the effort in 2012 as the second major contributor to the library. In 2015, pandas signed on as a fiscally sponsored project of NumFOCUS, a 501(c)(3) nonprofit charity in the United States.

TENSORFLOW

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Tensorflow is a symbolic math library based on dataflow and differentiable programming. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015. Starting in 2011, Google Brain built DistBelief as a proprietary machine learning system based on deep learning neural networks.

Its use grew rapidly across diverse Alphabet companies in both research and commercial applications.Google assigned multiple computer scientists, including Jeff Dean, to simplify and refactor the codebase of DistBelief into a faster, more robust application-grade library, which became TensorFlow.

In 2009, the team, led by Geoffrey Hinton, had implemented generalized backpropagation and other improvements which allowed generation of neural networks with substantially higher accuracy, for instance a 25% reduction in errors in speech recognition.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

In December 2017, developers from Google, Cisco, RedHat, CoreOS, and CaiCloud introduced Kubeflow at a conference. Kubeflow allows operation and deployment of TensorFlow on Kubernetes.In March 2018, Google announced TensorFlow.js version 1.0 for machine learning in JavaScript.

In Jan 2019, Google announced TensorFlow 2.0. It became officially available in Sep 2019. In May 2019, Google announced TensorFlow Graphics for deep learning in computer graphics.

KERAS

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.

As of version 2.4, only TensorFlow is supported. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet is also the author of the XCeption deep neural network model.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling. Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

PIL

Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7. Development of the original project, known as **PIL**, was discontinued in 2011. Subsequently, a successor project named **Pillow** forked the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu (since 13.04).

MATPLOTLIB

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter. Since then it has an active development community and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and was further joined by Thomas Caswell Matplotlib 2.0.x supports Python versions 2.7 through 3.10. Python 3 support started with Matplotlib 1.2. Matplotlib 1.4 is the last version to support Python 2.6. Matplotlib has pledged not to support Python 2 past 2020 by signing the Python 3 Statement.

LAYERS USED IN MODELS

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

output = activation(dot(input, kernel) + bias)

where,

- input represent the input data
- kernel represent the weight data
- dot represent numpy dot product of all input and its corresponding weights
- bias represent a biased value used in machine learning to optimize the model
- activation represent the activation function.

Let us consider sample input and weights as below and try to find the result −

- input as 2 x 2 matrix [[1, 2], [3, 4]]
- kernel as 2 x 2 matrix [[0.5, 0.75], [0.25, 0.5]]
- bias value as 0
- activation as linear. As we learned earlier, linear activation does nothing.

```
>>> import numpy as np

>>> input = [ [1, 2], [3, 4] ]
>>> kernel = [ [0.5, 0.75], [0.25, 0.5] ]
>>> result = np.dot(input, kernel)
>>> result array([[1. , 1.75], [2.5 , 4.25]])
>>>
```

result is the output and it will be passed into the next layer.

The output shape of the Dense layer will be affected by the number of neuron / units specified in the Dense layer. For example, if the input shape is **(8,)** and number of unit is 16, then the output shape is **(16,)**. All layer will have batch size as the first dimension and so, input shape will be represented by **(None, 8)** and the output shape as **(None, 16)**. Currently, batch size is None as it is not set. Batch size is usually set during training phase.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense

>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.input_shape
(None, 8)
>>> layer_1.output_shape
(None, 16)
>>>
```

where,

- layer_1.input_shape returns the input shape of the layer.
- layer_1.output_shape returns the output shape of the layer.

The argument supported by Dense layer is as follows −

- units represent the number of units and it affects the output layer.
- activation represents the activation function.
- use_bias represents whether the layer uses a bias vector.
- kernel_initializer represents the initializer to be used for kernel.
- bias_initializer represents the initializer to be used for the bias vector.
- kernel_regularizer represents the regularizer function to be applied to the kernel weights matrix.
- bias_regularizer represents the regularizer function to be applied to the bias vector.
- activity_regularizer represents the regularizer function tp be applied to the output of the layer.
- kernel_constraint represent constraint function to be applied to the kernel weights matrix.
- bias_constraint represent constraint function to be applied to the bias vector.

As you have seen, there is no argument available to specify the input_shape of the input data. input_shape is a special argument, which the layer will accept only if it is designed as first layer in the model.

Also, all Keras layer has few common methods and they are as follows −

get_weights

Fetch the full list of the weights used in the layer.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense
>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.get_weights()
>>> [array([[-0.19929028, 0.4162618 , 0.20081699,
   -0.25589502, 0.3612864 , 0.25088787, -0.47544873, 0.0321095 ,
   -0.26070702, -0.24102116, 0.32778358, 0.4667952 , -0.43322265,
   -0.14500427, 0.04341269, -0.34929228], [ 0.41898954,
0.42256463,
   0.2399621 , -0.272717 , -0.37069297, -0.37802136, 0.11428618,
0.12749982,
   0.10182762, 0.14897704, 0.06569374, 0.15424263, 0.42638576,
0.34037888, -0.15504825,
   -0.0740819 ], [-0.3132702 , 0.34885168, -0.3259498 , -
0.47076607, 0.33696914,
   -0.49143505, -0.04318619, -0.11252558, 0.29669464, -0.28431225,
-0.43165374,
   -0.49687648, 0.13632 , -0.21099591, -0.10608876, -0.13568914],
[-0.27421212,
   -0.180812 , 0.37240648,  0.25100648, -0.07199466, -0.23680925,
-0.21271884,
   -0.08706653, 0.4393121 , 0.23259485, 0.2616762 , 0.23966897, -
0.4502542 , 0.0058881
```

```
   , 0.14847124, 0.08835125], [-0.36905527, 0.08948278, -
0.19254792, 0.26783705,
   0.25979865, -0.46963632, 0.32761025, -0.25718856, 0.48987913,
0.3588251 ,
   -0.06586111, 0.2591269 , 0.48289275, 0.3368858 , -0.17145419, -
0.35674667],
   [-0.32851398, 0.42289603, -0.47025883, 0.29027188, -0.0498147 ,
0.46215963,
   -0.10123312, 0.23069787, 0.00844061, -0.11867595, -0.2602347 ,
   -0.27917898, 0.22910392, 0.18214619, -0.40857887, 0.2606709 ],
[-0.19066167,
   -0.11464512, -0.06768692, -0.21878994, -0.2573272 , 0.13698077,
0.45221198,
   0.10634196, 0.06784797, 0.07192957, 0.2946936 ,
   0.04968262, -0.15899467, 0.15757453, -0.1343019 , 0.24561536],
[-0.04272163,
   0.48315823, -0.13382411, 0.01752126, -0.1630218 , 0.4629662 , -
0.21412933,
   -0.1445911 , -0.03567278, -0.20948446, 0.15742278, 0.11139905,
0.11066687,
   0.17430818, 0.36413217, 0.19864106]], dtype=float32),
array([0., 0., 0., 0., 0., 0.,
   0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype = float32)]
>>>
```

- ***set_weights*** − Set the weights for the layer
- ***get_config*** − Get the complete configuration of the layer as an object which can be reloaded at any time.

config = layer_1.get_config()

from_config

Load the layer from the configuration object of the layer.

config = layer_1.get_config() reload_layer = Dense.from_config(config)

input_shape

Get the input shape, if only the layer has single node.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense
>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.get_weights()
>>> layer_1.input_shape
(None, 8)
```

input

Get the input data, if only the layer has single node.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense
>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
```

```
>>> model.add(layer_1)
>>> layer_1.get_weights()
>>> layer_1.input
<tf.Tensor 'dense_1_input:0' shape = (?, 8) dtype = float32>
```

- ***get_input_at*** − Get the input data at the specified index, if the layer has multiple node
- ***get_input_shape_at*** − Get the input shape at the specified index, if the layer has multiple node
- ***output_shape*** − Get the output shape, if only the layer has single node.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense
>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.get_weights()
>>> layer_1.output_shape (None, 16)
```

output

Get the output data, if only the layer has single node.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense
>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.get_weights()
>>> layer_1.output
<tf.Tensor 'dense_1/BiasAdd:0' shape = (?, 16) dtype = float32>
```

- ***get_output_at*** − Get the output data at the specified index, if the layer has multiple node
- ***get_output_shape_ at*** − Get the output shape at the specified index, if the layer has multiple node

DROP OUT LAYER

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training will be appropriately set to True automatically, and in other contexts, you can set the kwarg explicitly to True when calling the layer.

(This is in contrast to setting trainable=False for a Dropout layer. trainable does not affect the layer's behavior, as Dropout does not have any variables/weights that can be frozen during training.)

Arguments

- **rate**: Float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=(batch_size, 1, features).
- **seed**: A Python integer to use as random seed.

Call arguments

- **inputs**: Input tensor (of any rank).
- **training**: Python boolean indicating whether the layer should behave in training mode (adding dropout) or in inference mode (doing nothing).

Dropouts are the regularization technique that is used to prevent overfitting in the model. Dropouts are added to randomly switching some percentage of neurons of the network. When the neurons are switched off the incoming and outgoing connection to those neurons is also switched off. This is done to enhance the learning of the model. Dropouts are usually advised not to use after the convolution layers, they are mostly used after the dense layers of the network. It is always good to only switch off the neurons to 50%. If we switched off more than 50% then there can be chances when the model leaning would be poor and the predictions will not be good. Let us see how we can make use of dropouts and how to define them while building a CNN model. We will use the same MNIST data for the same.

FLATTEN LAYER

To bring all levels of a multi-layered image down to one plane. High-end graphics programs provide a multi-layer file format, such as the Photoshop Document (PSD), which enables elements in each layer to be manipulated independently. In order to save the layered image in a single-layer graphics format such as TIFF or JPEG, the image is said to be "flattened."

An Adobe PDF file is also flattened to remove a transparency layer when the document is rendered in a printer or in an application that does not support the additional layer.

See layers and PSD.

CONVOLUTION LAYER

In deep learning, a convolutional neural network (CNN or ConvNet) is a class of deep neural networks, that are typically used to recognize patterns present in images but they are also used for spatial data analysis, computer vision, natural language processing, signal processing, and various other purposes The architecture of a Convolutional Network resembles the connectivity pattern of neurons in the Human Brain and was inspired by the organization of the Visual Cortex. This specific type of Artificial Neural Network gets its name from one of the most important operations in the network: convolution.

What Is a Convolution?

Convolution is an orderly procedure where two sources of information are intertwined; it's an operation that changes a function into something else. Convolutions have been used for a long time typically in image processing to blur and sharpen images, but also to perform other operations. (e.g. enhance edges and emboss) CNNs enforce a local connectivity pattern between neurons of adjacent layers.

CNNs make use of filters (also known as kernels), to detect what features, such as edges, are present throughout an image. There are four main operations in a CNN:

- Convolution
- Non Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

The first layer of a Convolutional Neural Network is always a **Convolutional Layer.** Convolutional layers apply a convolution operation to the input, passing the result to the next layer. A convolution converts all the pixels in its receptive field into a single value. For example, if you would apply a convolution to an image, you will be decreasing

the image size as well as bringing all the information in the field together into a single pixel. The final output of the convolutional layer is a vector. Based on the type of problem we need to solve and on the kind of features we are looking to learn, we can use different kinds of convolutions.

BATCH NORMALIZATION LAYER

Batch normalization is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. The activations scale the input layer in normalization. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model. The layer is added to the sequential model to standardize the input or the outputs. It can be used at several points in between the layers of the model. It is often placed just after defining the sequential model and after the convolution and pooling layers. The below code shows how to define the BatchNormalization layer for the classification of handwritten digits. We will first import the required libraries and the dataset. Use the below code for the same.

WHAT IS ACTIVATION FUNCTION?

Simply put, an activation function is a function that is added into an artificial neural network in order to help the **network learn complex patterns in the data**. When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding **what is to be fired to the next neuron**. That is exactly what an activation function does in an ANN as well. **It takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell**. The comparison can be summarized in the figure below.

RELU RECTIFIED ACTIVATION FUNCTION

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

The function must also provide more sensitivity to the activation sum input and avoid easy saturation. The solution had been bouncing around in the field for some time, although was not highlighted until papers in 2009 and 2011 shone a light on it. The solution is to use the rectified linear activation function, or ReL for short.

A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU for short. Often, networks that use the rectifier function for the hidden layers are referred to as rectified networks. Adoption of ReLU may easily be considered one of the few milestones in the deep learning revolution, e.g. the techniques that now permit the routine development of very deep neural networks.

SYSTEM REQUIRMENTS

- **HARDWARE**
  - System : Intel i3/i5 2.4 GHz
  - Hard Disk : 500 GB
  - Ram : 4/8 GB

- **SOFTWARE**
  - Operating system : Windows XP/ Windows 7
  - Software Tool : Open CV Python
  - Coding Language : Python
  - Toolbox : Image processing toolbox

SYSTEM DESINGN/METHODOLOGY

The emergence of deep learning algorithms has resulted in highly complex cognitive tasks for computer vision and image recognition. Recently, deep learning models have become the most popular tool for big data analysis and Artificial intelligence, outperforming traditional image classification algorithms, and they are currently being down- scaled for feasible mobile implementation. The proposed deep learning model for bird image classification using the CNN framework is described as follows [20].

4.1 CNN ARCHITECUTRE

The model of CNN configuration for bird identification utilized a stack of convolutional layers comprising an input layer, two FC layers, and one final output softmax layer.

Each convolutional layer comprised (a) 5 X 5 convolution, (b) BN, (c) ReLU activation, and (d) Pooling layers.

This section explains how to construct an optimized CNN model, why the parameters and hyper parameters must be tuned before training, the total number of convolutional layers, the size of the kernels for all relative convolutional layers, and the likelihood of retaining the anode during dropout regularization for the dataset [15].

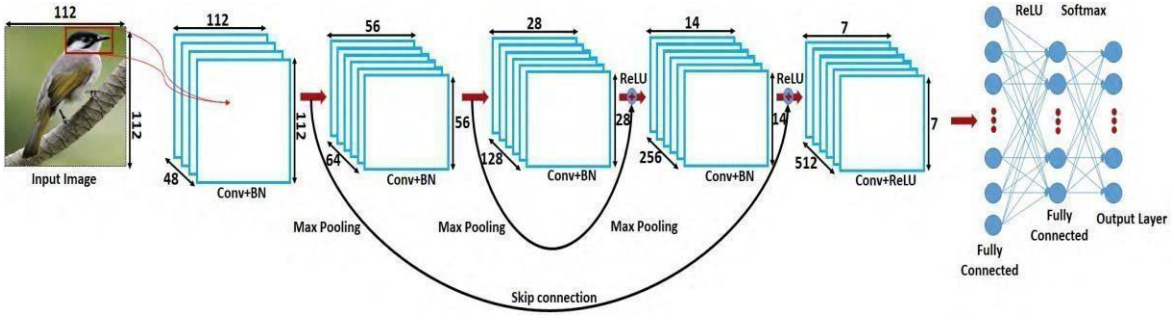


Figure 4.1 - CNN architecture for detecting bird images

4.2 ALGORITHM

Step 1: Image/video acquisition from the camera.

Step 2: Convert video to frames.

Step 3: Store images of each bird as database which is used as training set for our program

Step 4: Compare camera captured frames with the database.

Step 5: Use in read function to read the image and Preprocessing is done on that image. Perform Blob detection on the frame and blobs are matched with images from training data base images.

Step 6: And check if it is matching or not.

Step 7: To identification of that bird is desired or not. An array is created and program is written to classify the bird classes.

4.3 SKIP CONNECTIONS

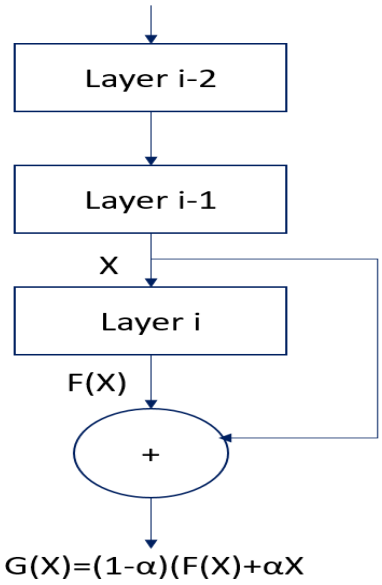


Figure 4.2 - Framework of skip connections

When images are learned, deep neural network models train a base network from scratch to identify associations of features and patterns in the target dataset.

Features are transformed from general to specific by the last layer of the network to predict the outputs of newly imposed inputs.

If first-layer features are general and last-layer features are specific, then transition from general to specific must have occurred somewhere in the network.

To address and quantify the degree to which a particular layer is general or specific, we proposed adding skip connections among corresponding convolutional layers, as shown in Fig. 4 [13]. The skip layer connections should improve feature extraction through weighted summation of corresponding layers as follows:

$$G(X) = (1 - \alpha)\, F(X) + \alpha X \qquad (1)$$

Where X is the input, $F(X)$ is a function of input X, $G(X)$ is a linear combination of $F(X)$ and X, and α is a weight in the unit interval [0, 1]. To check specific layers, we used different weights. For example, if $\alpha > 0.5$, then result from the previous layer contributes less to overall performance than the layers preceding it.

By contrast, if $\alpha < 0.5$, then result from the previous layer contributes more to the overall performance. Using these skip connections can facilitate network training by reducing Memory usage and increasing performance by concatenating the feature maps of each convolution layer.

4.4 TRAINING OF THE DATASET

- During training, input color images with a fixed size of 112 X 112 pixels were fed into CNN for feature extraction and bird image recognition.
- This study uses a dataset comprising 1879 images of 27 bird species. The dataset was split into 1749 images for training, 65 for validation, and 65 for testing.
- The input images passed through a hierarchical stack of convolutional layers to extract distinct features, such as color, shape, and edges, with varying orientations of the head, body, legs, and tail shown in the images.
- The first convolutional layer trans- formed the input image into pixels, propelled it to the next layer, and followed the feature extraction procedure until the input image had been precisely classified with a probability distribution [14].
- To capture the features of the input image, every convolutional filter had a kernel size of 3 X 3 pixels and a high activation map that slid across the entire input volume.
- The stride was fixed at one by shifting the kernel one unit at a time to control the filter convolving around the input of the next pixel so that the output volume would not shrink and the yield would be an integer rather than a fraction; that is, (i - k + 2q)/(s + 1), where i is the input height or length, k is the filter, q is the padding, and s is the stride.
- The padding was attuned to one around the input image to preserve the spatial resolution of output feature map after convolution; i.e. $q = (k - 1)/2$.
- Spatial pooling was implemented to localize and separate the chunks of images with a 2 X 2 pixel window size, max pooling, and two strides, where the maximum pixel rate in each chunk of an image was considered.
- The stack of convolutional layers was followed by an element-wise activation function, the ReLU, to maintain a constant volume throughout the network.
- To implement the skip connection in the network, down- sampling is performed by conv3 and conv4 with a stride of 2. We directly use skip connection when the input and output have the same dimensions.
- When the dimensions of the output are increased, the shortcut performs identity mapping with an extra zero-padding entry for increasing dimensions.

❖ Two FC layers were implemented with the same 4096-dimension configuration to learn the gradient descent, compute the target class scores in the training set for each image, and localize objects positioned anywhere in the input image. A schematic of The ConvNet architecture is presented in Fig. 2.

$$\text{softmax } \sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad \text{for } i = 1, \ldots, K. \qquad (2)$$

where x_i is the ith element of the input vector $\boldsymbol{x}$, $\sum_i \sigma(x)_i = 1$ and $\sigma(x)_i > 0$, which is the probability distribution over a set of outcomes.

4.5 FEATURE EXTRACTION

- Extracting features from raw input images is the primary task when extracting relevant and descriptive information for engrained object recognition.

- We separately extracted the features in relevant positions for each part of an image and subsequently learned the parts of the model features that were mapped directly To the corresponding parts.

- The features were calculated using ReLU 5 and ReLU 6.

- Localization was used to find object parts defined by bounding box coordinates and their dimensions (width and height) in the image.

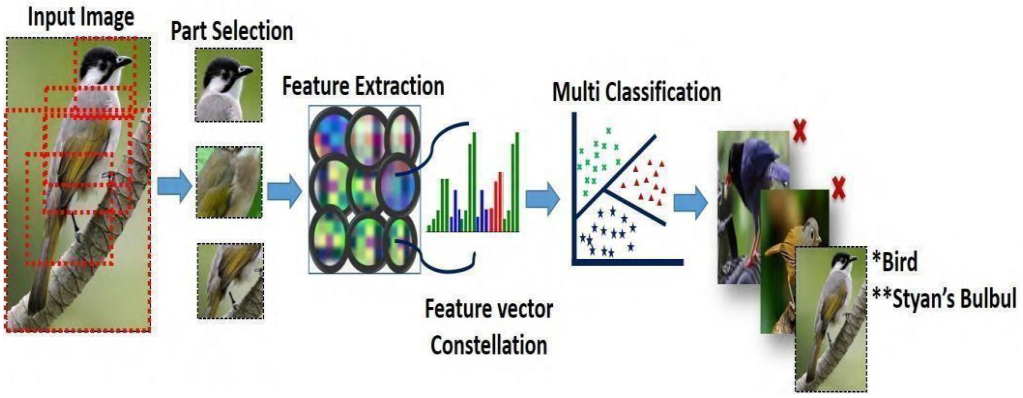


Figure 4.3 - Input raw data and feature illustration for a classifier

- Subsequent steps of the learning algorithm were for learning the map of the feature vectors of the input image, deciding whether the region fit an object class of interest, and then classifying the expected output with the correct labels in the image.

- For a given image, feature vectors represent the probability of target object centrality in the database, and the softmax classifier produces the probability scores for each label. Fig. 4 presents a raw input image, illustrating part selection and crucial feature identification. Multiclassification predicts a category label with the highest probability for the image [7].

IMPLEMENTATION

To complete the semantic bird search task, we established client–server architecture to bridge the communication gap between the cloud and mobile device over a network. The entire setup was executed in the following manner:-

- Raw bird images were distilled to remove irrelevant parts and learned by the CNN to yield parameters on the GPU platform. Subsequently, a TF inference model was developed in the workstation for deployment in the smartphone.
- The output was detected using a Google Colaboratory

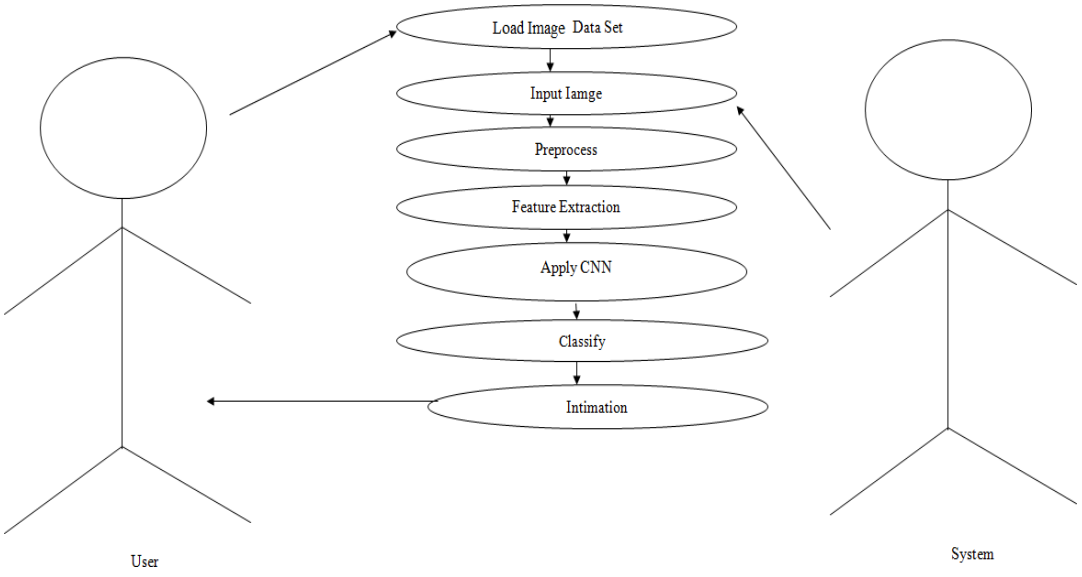


Figure 5.1 - Use Case Diagram

- As you can see in the use case diagram first the user loads the images in the dataset in a given directory.
- For training the model generally we have 3 different directories that are train, test and validations.
- In each directorythere will be sub directories of birds with their given set of images.

- Because of proper differentiation between each and every directories the training of the model does occur quiet efficiently.
- After the images is been set, the imagedatagenerator is being used to rescale each and every image in the scale of 1:255 pixel.
- After that feature extraction is being applied to predict each and every pixel after which the cumulative result is being passed on into CNN algorithm.
- For proper prediction CNN is used so as to get 4-5 possible probabilities of different features. Finally the highest probability feature is being used for further classification.
- Finally the actual prediction is being made out with a certain probability.

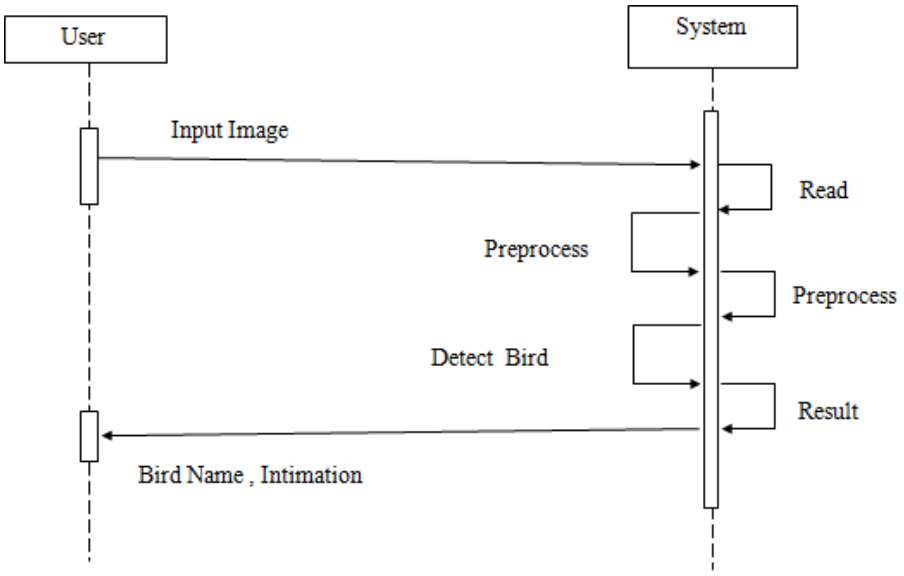


Figure 5.2 - Sequence Diagram

ImageDataGenerator

Keras **ImageDataGenerator** class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.

However, the main benefit ofusing the Keras ImageDataGenerator class is that it is designed to provide real-time data augmentation. Meaning it is generating augmented images on the fly while your model is still in the training stage.

ImageDataGenerator class ensures that the model receives new variations of the images at each epoch. But it only returns the transformed images and does not add it to the original corpus of images. If it was, in fact, the case, then the model would be seeing the original images multiple times which would definitely overfit our model.

Another advantage of ImageDataGenerator is that it requires lower memory usage. This is so because without using this class, we load all the images at once. But on using it, we are loading the images in batches which saves a lot of memory.

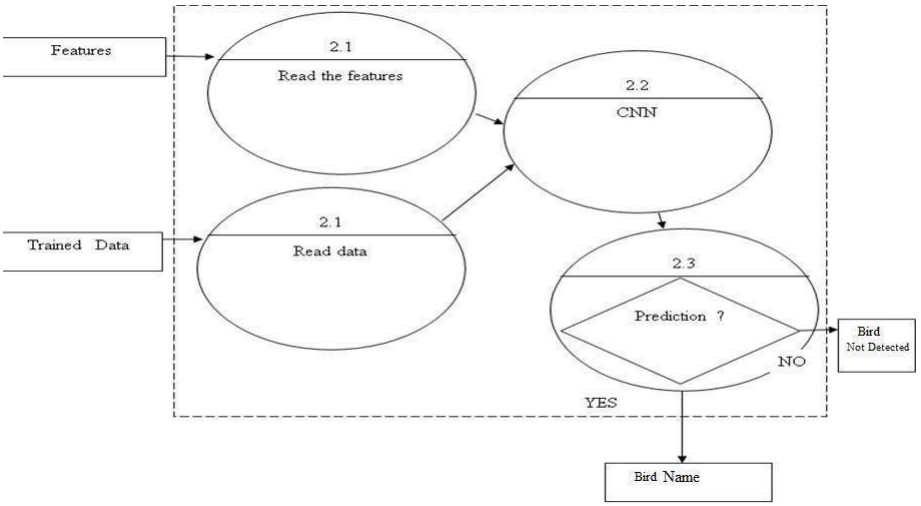


Figure 5.3 - Dataflow Diagram

ResNet50V2

ResNet50V2 is a modified version of ResNet50 that performs better than ResNet50 and ResNet101 on the ImageNet dataset. In **ResNet50V2**, a modification was made in the propagation formulation of the connections between blocks. **ResNet50V2** also achieves a good result on the ImageNet dataset.

RMSprop

RMSprop is a gradient-based optimization technique used in training neural networks. It was proposed by the father of back-propagation, Geoffrey Hinton. Gradients of very complex functions like neural networks have a tendency to either vanish or explode as the data propagates through the function (refer to vanishing gradients problem). Rmsprop was developed as a stochastic technique for mini-batch learning.

RMSprop deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing.

Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyper parameter. This means that the learning rate changes over time.

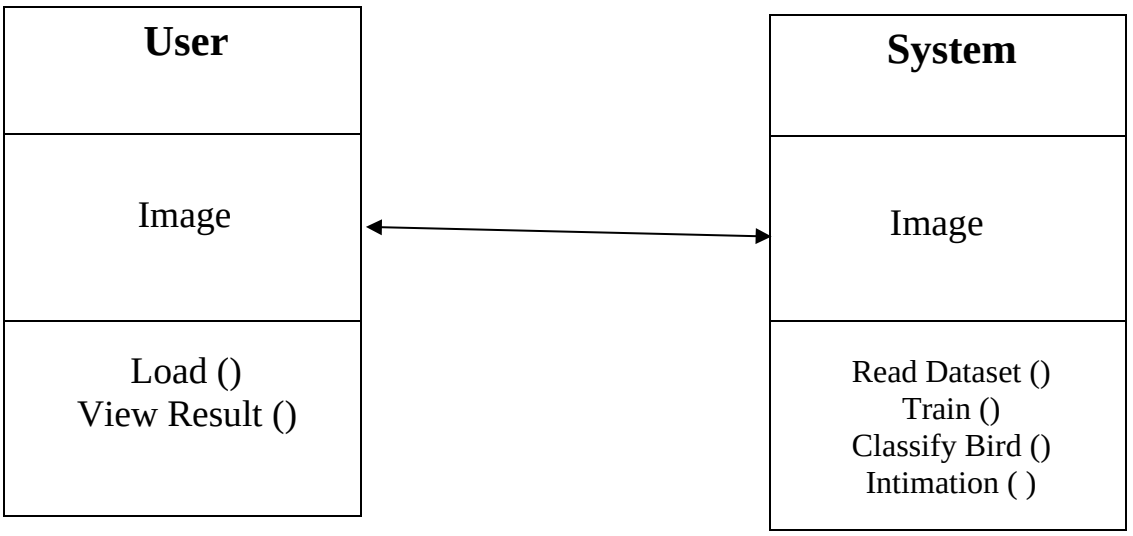


Figure 5.4 - Class Diagram

As you can see in the above class diagram first user loads the image into the dataset. After that the system reads the data and train the model based on the data. As the model is trained we can directly check out the final result.

In this subsection, we explain using a high-resolution smart- phone camera to identify and classify bird information based on deep learning. To complete the semantic bird search task, we established client–server architecture to bridge the communication gap between the cloud and mobile device over a network. The entire setup was executed in the following manner:

- Raw bird images were distilled to remove irrelevant parts and learned by the CNN to yield parameters on the GPU platform. Subsequently, a TF inference model was developed in the workstation for deployment in the smartphone.
- The output was detected using an Android app platform or through the web.

On the workstation/server side, the following segments were considered. The TF backend session model for object detection was prepared to save the TF computation graphs of input, output, weight, and bias as graph_def text files (tfdroid.pbtxt), which comprised the entire architecture of the model. The CNN architecture was trained to load the raw input data of bird images using Keras callbacks with the predefined parameters into TF format to fit the model for inference.

After training the model, the parameters of all object was created for the session, and the checkpoints, model name, model path, and input–output parameter layers of the model were defined.

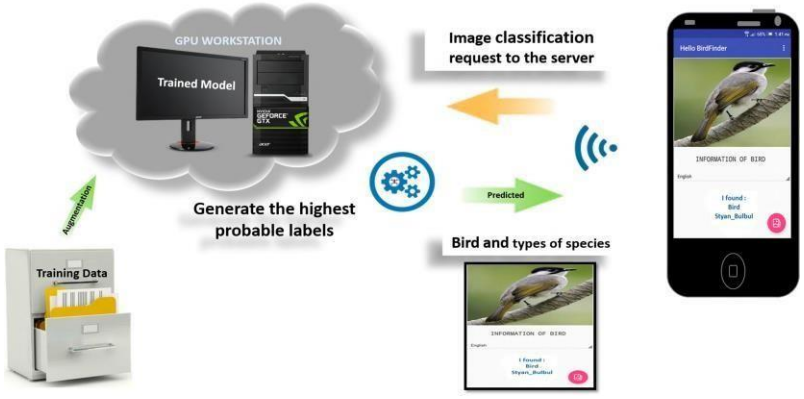


Figure 5.5 – Client Server Architecture for Bird Detection

All other explicit metadata assignments that were not necessary for the client–server inference, such as GPU directories on the graph nodes or graph paths, were removed. In this bird detection model, the output layer pro- vides: (a) the parts of the input image containing a bird, (b) type of bird species, and (c) parts of the input image not containing a bird. Finally, the trained model was frozen by converting all variable parameters in the checkpoint file into constants (stops). Subsequently, both files were serialized into a single file as a ProtoBuf graph_def.

The graph frozen as a ProtoBuf graph_def can be optimized, if required, for feasibility inference. The saved ProtoBuf graph_def was reloaded and resaved to a serialized string value. The following actions were considered when optimizing for inference:

- Removal of redundant variables
- Stripping out of unused nodes
- Compression of multiple nodes and expressions into a distinct node
- Removal of debug operations, such as CheckNumerics, that is not necessary for inference
- Group batch norm operation into precalculated weights
- Fusing of common operations into unified versions
- Reduction of model size through quantization and weight rounding
- Fixing of hardware assignment problems

Once the model was trained and saved for mobile inference in the workstation, we created an Android app to copy and configure the TF inference files. On the client/mobile side, the SDK written in Java and NDK written in C were downloaded to create mobile interface activities and to communicate with the pretrained CNN TF ProtoBuf files that contained the model definition parameters and weights.

The JNI was used to bridge the TF and Android platforms. The JNI executes the load Model function and obtains predictions of an object from the TF ProtoBuf files using the Android NDK. After classifying the object in the pertained model, the classified label output is sent back to the mobile phone using the Android NDK.

Using the aforementioned client–server computing setup, we provided a mechanism to encapsulate the cloud and mobile session. Bird recognition can be executed through cloud- and device-based inference. In this approach to deep learning inference on a mobile device, the trained model parameters are loaded into the mobile app, and the computations are completed locally on the device to predict the image output. The mobile phone is constrained by memory size and inflexibility when updating the trained model [14].

However, in the cloud-based deep learning model, the trained model is stored on a remote server, and the server connects to the mobile device via the Internet using a web API to predict The uploaded images. Therefore, deploying the learned architecture with the cloud-based model can be easily ported to various platforms or mobile phones, and can upscale the model with new features without much difficulty. Because of the aforementioned benefits, cloud-based inference was used to execute bird image recognition [15].

Fig. 9 shows the proposed system for bird information retrieval from the trained model stored in the workstation. The server with the TF platform takes prediction requests for bird images from client mobile phones and feeds and processes in the deep learning trained model the images sent from the API. After an image has been predicted, the TF platform classifies and generates the probability distribution of the image and transmits the query image result back to the user's mobile phone with the classified label [14].

To analyse the uploaded images, we used a mobile phone as a client to perform the following functions: the end-user interface captures the bird image and instantly or directly uploads the image from the gallery of the mobile phone to extract image features. The mobile app sends an HTTPS request to the web server (central computer system) to retrieve the pretrained database regarding the uploaded bird image. The server performs data aggregation and an exhaustive search using the uploaded image to determine the matching parameters and retrieve information related to the images [13].

To optimize binary segmentation of the weighted graph of the image, Grabcut semantic foreground segmentation is applied for bird species categorization. The head of a bird is the main prior-fitted region of interest, the other parts of the bird are lower priority regions of interest [14].

A Colour model is projected to filter the original image with the bounding box. Subsequently, the information is classified and mapped, and the correctness of the matched image is transmitted back to the user's mobile phone. The transmitted file contains metadata related to the bird's information with the classified label indicating a bird species. Fig. 9 shows the interface steps of bird detection.

TESTING AND RESULTS

The pro- posed system can predict and differentiate bird and nonbird images.

To acquire the output of images with or without birds, the multiscale sliding window strategy was applied so that the extracted sub window could define the target object.

```
optimizer=keras.optimizers.RMSprop(learning_rate=0.0001)
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
history=model.fit(train_generator,epochs=7,validation_data=val_generator)
```

```
Epoch 1/7
7/7 [==============================] - 467s 60s/step - loss: 1.3182 - accuracy: 0.6238 - val_loss: 0.0238 - val_accuracy: 1.0000
Epoch 2/7
7/7 [==============================] - 300s 43s/step - loss: 0.2059 - accuracy: 0.9657 - val_loss: 0.0067 - val_accuracy: 1.0000
Epoch 3/7
7/7 [==============================] - 307s 44s/step - loss: 0.0986 - accuracy: 0.9874 - val_loss: 0.0095 - val_accuracy: 1.0000
Epoch 4/7
7/7 [==============================] - 302s 43s/step - loss: 0.0519 - accuracy: 0.9971 - val_loss: 0.0046 - val_accuracy: 1.0000
Epoch 5/7
7/7 [==============================] - 311s 45s/step - loss: 0.0318 - accuracy: 0.9994 - val_loss: 0.0066 - val_accuracy: 1.0000
Epoch 6/7
7/7 [==============================] - 302s 43s/step - loss: 0.0239 - accuracy: 0.9983 - val_loss: 0.0062 - val_accuracy: 1.0000
Epoch 7/7
7/7 [==============================] - 310s 44s/step - loss: 0.0127 - accuracy: 0.9994 - val_loss: 0.0051 - val_accuracy: 1.0000
```

In the above demonstration as you increase the epoch value that was set to 7, it will increase the time to train the model. However higher value of epoch allow the machine to learn even better and the final result will be even better. Also you can see the corresponding time taken to the epoch number. With time you will also notice decrease in loss value and increase in accuracy as the machine gets better with time.

Below are the following two test cases that were demonstrated –

S1 #Test Case: -	**UTC-1**
Name of Test: -	**Image Capture and Bird Detection**
Items being tested: -	**Bird Detection**
Sample Input: -	**Image**
Expected output: -	**Species of Bird should be detected**
Actual output: -	**Species of Bird is done successfully**
Remarks: -	**Pass**

S2 #Test Case: -	UTC-2
Name of Test: -	Image capture of random object other than bird
Items being tested: -	Bird Detection
Sample Input: -	Image
Expected output: -	Random Species of bird with low probability should be detected
Actual output: -	Random Species of bird with low probability
Remarks: -	Pass

The base learning rate was 0.01 and subsequently shifted to 0.0001. The network was trained until the cross-entropy stabilized. Skip connections were implemented when the input and output layers had equal weights. For instance, when the dimensions of the output were increased, the weights were concatenated in a deeper layer to capture and reconstruct features more effectively in the next layer.

```
import pandas as pd
df = pd.DataFrame(history.history )
print(df)
```

```
       loss  accuracy  val_loss  val_accuracy
0  1.318174  0.623785  0.023835           1.0
1  0.205859  0.965695  0.006744           1.0
2  0.098582  0.987421  0.009502           1.0
3  0.051870  0.997141  0.004594           1.0
4  0.031799  0.999428  0.006575           1.0
5  0.023901  0.998285  0.006163           1.0
6  0.012739  0.999428  0.005112           1.0
```

Figure 6.1 - Loss-Accuracy Data

This studydealt predominantly for bird recognition with 1749 images for training, 65 images for testing and 65 for validation. The proposed system could detect and differentiate uploaded images as birds with an overall accuracy of 98.70% for the training dataset.

The Below figure suggest us the accuracy score for each and every epoch value –

```
plt.plot(history.history['accuracy'],c='b',label='train')
plt.plot(history.history['val_accuracy'],c='r',label='validation')
plt.legend(loc='lower right')
plt.show()
```

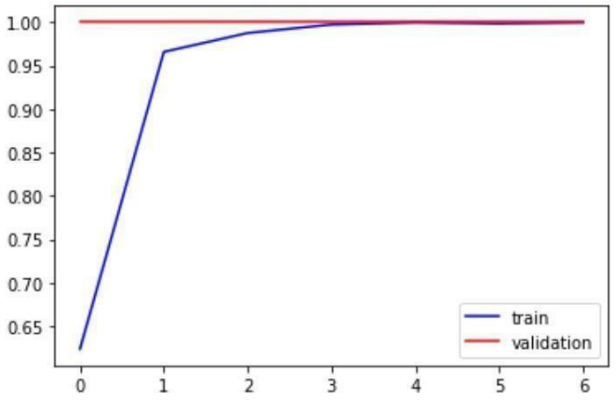


Figure 6.2 - Accuracy data for train and validation

The Below figure suggest us the loss score for each and every epoch value –

```
plt.plot(history.history['loss'],c='b',label='train')
plt.plot(history.history['val_loss'],c='r',label='validation')
plt.legend(loc='upper right')
plt.show()
```

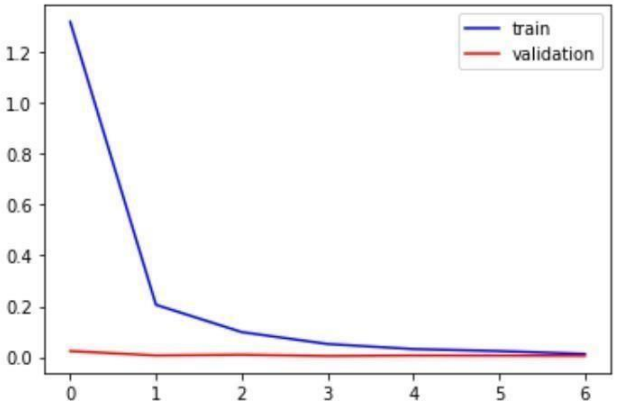


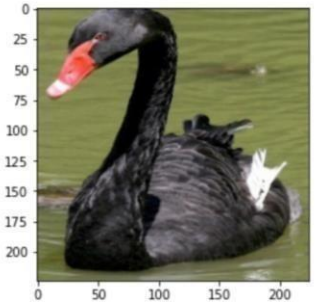
Figure 6.3 - Loss data for train and validation

NOTE – An important point that we need to remember is the above data will vary every time we train the model. Also with change in epoch value and learning rate, the above data will also change.

The Results are as follows –

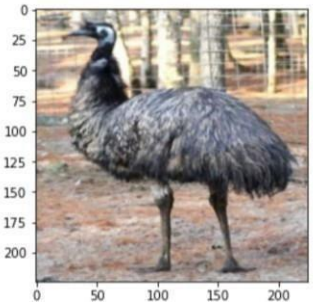
```
predict('/content/drive/MyDrive/Colab Notebooks/bird/test/BLACK SWAN/2.jpg')
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_
  warnings.warn('`model.predict_classes()` is deprecated and '
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:430: UserWarning: `model.predict_
  warnings.warn('`model.predict_proba()` is deprecated and '
The predicted image of the bird is: BLACK SWAN with a probability of 100.0%
```



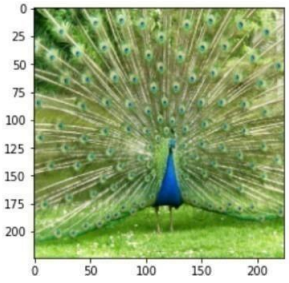

```
predict('/content/drive/MyDrive/Colab Notebooks/bird/test/EMU/5.jpg')
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_
  warnings.warn('`model.predict_classes()` is deprecated and '
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:430: UserWarning: `model.predict_
  warnings.warn('`model.predict_proba()` is deprecated and '
The predicted image of the bird is: EMU with a probability of 100.0%
```




```
predict('/content/drive/MyDrive/Colab Notebooks/bird/test/PEACOCK/5.jpg')
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_
  warnings.warn('`model.predict_classes()` is deprecated and '
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:430: UserWarning: `model.predict_
  warnings.warn('`model.predict_proba()` is deprecated and '
The predicted image of the bird is: PEACOCK with a probability of 100.0%
```



CONCLUSION AND FUTURE WORKS

This study developed a prediction platform that uses cloud- based deep learning for image processing to identify bird species from digital images uploaded by an end-user.

This study dealt predominantly for bird recognition with 1749 images for training, 65 images for testing and 65 for validation.

The proposed system could detect and differentiate uploaded images as birds with an overall accuracy of 98.70% for the training dataset.

This study ultimately aimed to design an automatic system for differentiating fine-grained objects among bird images with shared fundamental characteristics but minor variations in appearance.

However in this study, we are more focused on predicting the 27 species of bird endemic to Taiwan more efficient and effective. The proposed model can predict the uploaded image of a bird as bird with 100% accuracy.

But due to the subtle visual similarities between and among the bird species, the model sometimes lacks the interspecific comparisons and among the bird species and eventually leads to comparisons and among the bird species and eventually leads to misclassification. In average, the test dataset yields 93.79% of sensitivity, 96.11% of specificity and this model can be used for prediction and classification of the endemic bird images.

In the future, we intend to develop a method for predicting different generations of specific bird species within the intraclass and interclass variations of birds and to add more bird species to our database.

The proposed architecture encountered some limitations and has room for improvement in the future. Sometime the model confused the prediction of endemic birds when the uploaded bird images shared similar colors and size. If most bird species within a district need to be retrieved from the system, the database must be updated and need to be retrained with new features of the birds.

For extending the proposed system to some specific districts for birdwatching may encounter imbalanced distribution of the dataset among the bird species if only a small size of dataset is available.

In the future, we intend to develop a method for predicting different generations of specific bird species within the intraclass and interclass variations of birds and to expand bird species to our database so that more people can admire the beauty from watching birds.

REFERENCES

[1] D. T. C. Cox and K. J. Gaston, ‘‘Likeability of garden birds: Importance of species knowledge & richness in connecting people to nature,’’ *PloS one,* vol. 10, no. 11, Nov. 2015, Art. no. e0141505.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, ‘‘ImageNet large scale visual recognition challenge,’’ *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[3] H. Yao, S. Zhang, Y. Zhang, J. Li, and Q. Tian, ‘‘Coarse-to-fine description for fine-grained visual categorization,’’ *IEEE Trans. Image Process.*, vol. 25, no. 10, pp. 4858–4872, Oct. 2016.

[4] F. Garcia, J. Cervantes, A. Lopez, and M. Alvarado, ‘‘Fruit classification by extracting color chromaticity, shape and texture features: Towards an application for supermarkets,’’ *IEEE Latin Amer. Trans.,* vol. 14, no. 7, pp. 3434–3443, Jul. 2016.

[5] L. Zhu, J. Shen, H. Jin, L. Xie, and R. Zheng, ‘‘Landmark classification with hierarchical multi-modal exemplar feature,’’ *IEEE Trans. Multimedia*, vol. 17, no. 7, pp. 981–993, Jul. 2015.

[6] X. Liang, L. Lin, W. Yang, P. Luo, J. Huang, and S. Yan, ‘‘Clothes co-parsing via joint image segmentation and labeling with application to clothing retrieval,’’ *IEEE Trans. Multimedia*, vol. 18, no. 6, pp. 1175–1186, Jun. 2016.

[7] Y.-P. Huang, L. Sithole, and T.-T. Lee, ‘‘Structure from motion technique for scene detection using autonomous drone navigation,’’ *IEEE Trans. Syst., Man, Cybern. Syst.,* to be published.

[8] C. McCool, I. Sa, F. Dayoub, C. Lehnert, T. Perez, and B. Upcroft, ‘‘Visual detection of occluded crop: For automated harvesting,’’ in *Proc. Int. Conf. Robot. Autom. (ICRA)*, Stockholm, Sweden, May 2016, pp. 2506–2512.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘‘ImageNet classification with deep convolutional neural networks,’’ in *Proc. 25th Int. Conf. Advance Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2012, pp. 1097–1105.

[10] B. Zhao, J. Feng, X. Wu, and S. Yan ‘‘A survey on deep learning-based fine- grained object classification and semantic segmentation,’’ *Int. J. Automat. Comput.* vol. 14, no. 2, pp. 119–135, Apr. 2017.

[11] H. Yang, J. T. Zhou, Y. Zhang, B.-B. GAO, J. Wu, and J. CAI, ‘‘Exploit bounding box annotations for multi-label object recognition,’’ in *Proc. Int. Conf. Comput. Vision Pattern Recognit.* Las Vegas, NV, USA, Jun. 2016, pp. 280–288.

[12] Li Liu, W. Ouyang, X. Wang, P. Fieguth, X. Liu, and M. Pietikäinen, ‘‘Deep learning for generic object detection: A survey,’’ Sep. 2018, *arXiv:1809.02165*. [Online]. Available: https://arxiv.org/abs/1809.02165

[13] K. Dhindsa, K. D. Gauder, K. A. Marszalek, B. Terpou, and S. Becker, ‘‘Progressive thresholding: Shaping and specificity in automated neuro- feedback training,’’ *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 26, no. 12, pp. 2297–2305, Dec. 2018.

[14] C.-Y. Lee, A. Bhardwaj, W. Di, V. Jagadeesh, and R. Piramuthu, ‘‘Region- based discriminative feature pooling for scene text recognition,’’ in *Proc. Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 4050–4057.

[15] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, ‘‘Simultaneous detection and segmentation,’’ in *Proc. Eur. Conf. Comput. Vis.*, Jul. 2014, pp. 297–312.

[16] S. Branson, G. V. Horn, S. Belongie, and P. Perona, ‘‘Bird species categorization using pose normalized deep convolutional nets,’’ in. *Proc. Brit. Mach. Vis. Conf.*, Nottingham, U.K., Jun. 2014, pp. 1–14.

[17] B. Yao, A. Khosla, and L. Fei-Fei, ‘‘Combining randomization and discrimination for fine-grained image categorization,’’ in *Proc. CVPR*, Colorado Springs, CO, USA, USA, Jun. 2011, pp. 1577–1584.

[18] Y.-B. Lin, Y.-W. Lin, C.-M. Huang, C.-Y. Chih, and P. Lin, ‘‘IoTtalk: A management platform for reconfigurable sensor devices,’’ *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1552–1562, Oct. 2017.

[19] X. Zhang, H. Xiong, W. Zhou, and Q. Tian, ‘‘Fused one-vs-all features with semantic alignments for fine-grained visual categorization,’’ *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 878–892, Feb. 2016.

[20] Google Android Developer. *Render Script API Guides*. Accessed: Nov. 20, 2018. [Online]. Available:https://developer.android.com/guide/topics/renderscript/compute.html

[21] M. Z. Andrew and G. Howard. (Jun. 14, 2017), *MobileNets: Open- Source Models for Efficient On-Device Vision*. Accessed: Feb. 20, 2018. [Online]. Available: https://research.googleblog.com/2017/06/mobilenets- open-source models-for.html

[22] C. Koylu, C. Zhao, and W. Shao, ‘‘deep neural networks and kernel density estimation

for detecting human activity patterns from Geo-tagged images: A case studyof birdwatching on Flickr,'' *Int. J. Geo-Inf.*, vol. 8, no. 1, p. 45, Jan. 2019.

[23] R. Richa, R. Linhares, E. Comunello, A. von Wangenheim, J.-Y. Schnitzler, B. Wassmer, C. Guillemot, G. Thuret, P. Gain, G. Hager, and R. Taylor, ''Fundus image mosaicking for information augmentation in computer- assisted slit-lamp imaging,'' *IEEE Trans. Med. Imag.*, vol. 33, no. 6, pp. 1304–1312, Jun. 2014.

[24] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, ''Deep con- volutional neural network architecture with reconfigurable computation patterns,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[25] P. Wang, W. Li, Z. Gao, J. Zhang, C. Tang, and P. O. Ogunbona, ''Action recognition from depth maps using deep convolutional neural networks,'' *IEEE Trans. Human–Mach. Syst.*, vol. 46, no. 4, pp. 498–509, Aug. 2016.

[26] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, ''Stacked convolutional