Configuration Manual

MSc Research Project
MSc. in Data Analytics

Nawaz Sheikh
Student ID: x18134637

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nawaz Sheikh
Student ID:	x18134637
Programme:	MSc. in Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	12th December 2019
Project Title:	Identification and Classification of Wildlife from Camera-Trap Images using Machine Learning and Computer Vision
Word Count:	1223
Page Count:	38

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	12th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nawaz Sheikh
x18134637

1 Introduction

Most of the system setup, hardware and software requirements and the implementation and evaluation along with exploratory data analaysis has been explained in this configuration manual.

2 Exploratory Data Analysis

2.1 Mount the Google Drive

Google Drive is mounted so that the files can be accessed.[1]

```
[ ] #Mount the google drive:
    from google.colab import drive
    drive.mount('/content/drive')

    Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6

    Enter your authorization code:
    ··········
    Mounted at /content/drive
```

Figure 1: Google Drive

```
[ ] animals_path = "/content/drive/My Drive/Colab Notebooks/Six_Classes"
```

Figure 2: Folder directory

The folder directory is set where the dataset is present so that files can be accessed from the directory mentioned.

[1] https://drive.google.com/drive/folders/1Q6kJBWvVC$_D dSAvg97jp65Cs3KoUjbYO$

2.2 Importing the libraries

```
#Importing the libraries:
import tensorflow
import keras
import imageai
```

```
import os
import shutil
import glob
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import math
import re
import sys
```

Figure 3: Importing libraries

Libraries of tensorflow, keras, glob, matplotlib, math are imported so that methods and functions are used later on for certain tasks.[2]

2.3 Distribution of selected classes

```
import pygal
from IPython.display import display, HTML
#Create function to display interactive plotting
base_html = """
<!DOCTYPE html>
<html>
  <head>
  <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.jquery.js"></script>
  <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pygal-tooltips.min.js""></script>
  </head>
  <body>
    <figure>
      {rendered_chart}
    </figure>
  </body>
</html>
"""

def galplot(chart):
    rendered_chart = chart.render(is_unicode=True)
    plot_html = base_html.format(rendered_chart=rendered_chart)
    display(HTML(plot_html))

#Compare class distribution
line_chart = pygal.Bar(height=300)
line_chart.title = 'Animals Distribution'
for o in os.listdir(animals_path):
    line_chart.add(o, len(os.listdir(os.path.join(animals_path, o))))
galplot(line_chart)
```

Figure 4: Setting pygal library

[2]https://www.tensorflow.org/

Pygal library is used to plot the available classes. The plot is done using bar graph. A wrapper is created to render the chart inline and data is passed through it so that it can be displayed.

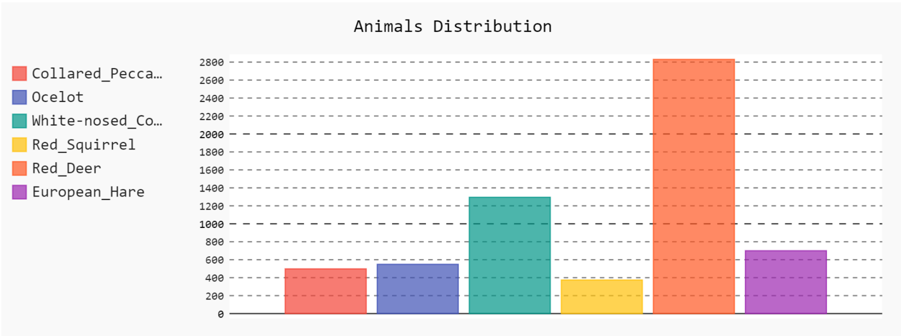

Figure 5: Class distribution

Figure 5 refers to the class distribution obtained for the classes present in the original dataset. It seems that Red Deer has the highest number of samples making it the majority class and Red Squirrel has the lowest number of samples making it the minority class as compared to other 5 classes.

2.4 Confirm Folder Structure

```
#Confirm Folder Structure
for root, dirs, files in os.walk(animals_path):
    level = root.replace(os.getcwd(), '').count(os.sep)
    print('{0}{1}/'.format('    ' * level, os.path.basename(root)))
    for f in files[:2]:
        print('{0}{1}'.format('    ' * (level + 1), f))
    if level is not 0:
        print('{0}{1}'.format('    ' * (level + 1), "..."))
```

```
Six_Classes/
    Collared_Peccary/
        SEQ88200_IMG_0003.JPG
        SEQ88200_IMG_0007.JPG
        ...
    Ocelot/
        SEQ75294_IMG_0005.JPG
        SEQ75294_IMG_0003.JPG
        ...
    White-nosed_Coati/
        SEQ84536_IMG_0001.JPG
        SEQ84536_IMG_0008.JPG
        ...
    Red_Squirrel/
        SEQ75972_IMG_0002.JPG
        SEQ76082_IMG_0001.JPG
        ...
    Red_Deer/
        SEQ80452_IMG_0016.JPG
        SEQ80452_IMG_0019.JPG
        ...
    European_Hare/
        SEQ75140_IMG_0004.JPG
        SEQ75140_IMG_0001.JPG
        ...
```

Figure 6: Folder Structure

Here, the folder structure is verified so that we can go ahead with applying models to the dataset. The output shows that there are six folders for six different classes. In each folder, there are sequences of different images.

2.5 Create Train, Validation and Test folders

```
[ ] import math
    import re
    import sys

    #Train and Test Set Variables
    train_val_test_ratio = (.7,.1,.2) # 70/10/20 Data Split
    test_folder = 'test/'
    train_folder = 'train/'
    val_folder = 'val/'

    file_names = os.listdir('/content/drive/My Drive/Colab Notebooks/Six_Classes')

    #Remove Existing Folders if they exist
    for folder in [test_folder, train_folder, val_folder]:
        if os.path.exists(folder) and os.path.isdir(folder):
            shutil.rmtree(folder)

    #Remake Category Folders in both Train and Test Folders
    for category in file_names:
        os.makedirs(test_folder + category)
        os.makedirs(train_folder + category)
        os.makedirs(val_folder + category)
```

Figure 7: Creation of training, validation and test folders

```
#Split Data by Train Ratio and copy files to correct directory
for idx, category in enumerate(file_names):
    file_list = os.listdir(animals_path + '/' + category)

    train_ratio = math.floor(len(file_list) * train_val_test_ratio[0])
    val_ratio = math.floor(len(file_list) * train_val_test_ratio[1])
    train_list = file_list[:train_ratio]
    val_list = file_list[train_ratio:train_ratio + val_ratio]
    test_list = file_list[train_ratio + val_ratio:]

    for i, file in enumerate(train_list):
        shutil.copy(animals_path + '/' + category + '/' + file, train_folder + '/' + category + '/' + file)
    sys.stdout.write('Moving %s train images to category folder %s' % (len(train_list), category))
    sys.stdout.write('\n')
    for i, file in enumerate(val_list):
        shutil.copy(animals_path + '/' + category + '/' + file, val_folder + '/' + category + '/' + file)
    sys.stdout.write('Moving %s validation images to category folder %s' % (len(val_list), category))
    sys.stdout.write('\n')
    for i, file in enumerate(test_list):
        shutil.copy(animals_path + '/' + category + '/' + file, test_folder + '/' + category + '/' + file)
    sys.stdout.write('Moving %s test images to category folder %s' % (len(test_list), category))
    sys.stdout.write('\n')

print("Done.")
```

Figure 8: Data split

The idea is to create three folders, train, validation and test with the data split ratio of 70%, 10% and 20% respectively. The directory where the folders should be created is listed. Also, if there is train, validate and test folder already existing in the listed directory, then they are removed and the folders are recreated after then the data is split in the consecutive folders.

```
Moving 348 train images to category folder Collared_Peccary
Moving 49 validation images to category folder Collared_Peccary
Moving 101 test images to category folder Collared_Peccary
Moving 384 train images to category folder Ocelot
Moving 54 validation images to category folder Ocelot
Moving 111 test images to category folder Ocelot
Moving 906 train images to category folder White-nosed_Coati
Moving 129 validation images to category folder White-nosed_Coati
Moving 260 test images to category folder White-nosed_Coati
Moving 261 train images to category folder Red_Squirrel
Moving 37 validation images to category folder Red_Squirrel
Moving 76 test images to category folder Red_Squirrel
Moving 1980 train images to category folder Red_Deer
Moving 283 validation images to category folder Red_Deer
Moving 567 test images to category folder Red_Deer
Moving 489 train images to category folder European_Hare
Moving 70 validation images to category folder European_Hare
Moving 141 test images to category folder European_Hare
Done.
```

Figure 9: Output for data split

2.6 Data Augmentation

This step is carried out to tackle the issue of class imbalance. Minority classes are augmented with a shift of 45 degrees so that the number of images for the minority class to train increases.

A check is carried out after the first 45 degrees shift of data augmentation to see the increase in the number of images for the minority class. The minority classes are again augmented to balance the classes.

Firstly, an example image is taken to observe the effect of augmentation. Then the image is displayed. A number of 4 augmentations is done to each image of the minority class. The augmentation is completed randomly on the minority class images. It is carried out for the training data only in order to stop the class bias. The validation and testing images remains the same as passes on originally after segregation.

```
[ ] import random
    import numpy as np
    from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

Using TensorFlow backend.
```

Figure 10: Importing the library for data augmentation

```
[ ] #Select a random image and follow the next step
    datagen = ImageDataGenerator(rotation_range=45,
                                 width_shift_range=0.2,
                                 height_shift_range=0.2,
                                 zoom_range=0.3,
                                 vertical_flip=True,
                                 horizontal_flip=True,
                                 fill_mode="nearest")
```

Figure 11: Image Data Generator

```
[ ] #Load example image
    file_list = glob.glob("test/*/*")
    img_path = random.choice(file_list)
    img = load_img(img_path)
    animal_class = img_path.split("/")[1]
    plt.imshow(img)
    plt.axis("off")
    plt.title("Original " + animal_class, fontsize=16)
```

```
Text(0.5, 1.0, 'Original 1.03-Collared_Peccary')
```

Figure 12: Sample image

```
img = img_to_array(img)
img = img.reshape((1,) + img.shape)
```

Figure 13: Image to array

```
#Apply different augmentation techniques
n_augmentations = 4
plt.figure(figsize=(15, 6))
i = 0
for batch in datagen.flow(img,
                          batch_size=1,
                          seed=21):

    plt.subplot(2, int(np.ceil(n_augmentations * 1. / 2)), i + 1)
    plt.imshow(array_to_img(batch[0]))
    plt.axis("off")
    plt.suptitle("Augmented " + animal_class, fontsize=16)

    i += 1
    if i >= n_augmentations:
        break
```

Figure 14: Augmentation technique

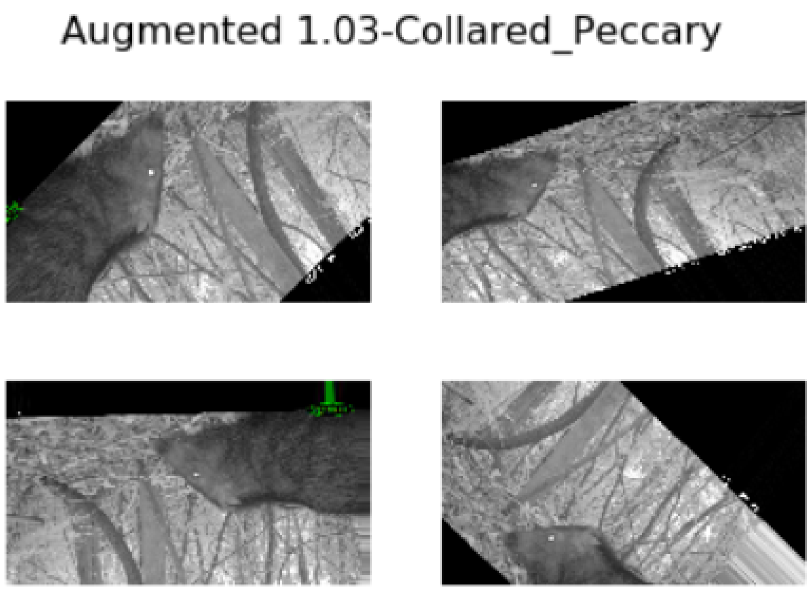

Figure 15: Augmentation example

```
[ ] #Oversampling Minority Classes in Training Set
    def data_augment(data_dir):
        list_of_images = os.listdir(data_dir)
        datagen = ImageDataGenerator(rotation_range=45,
            horizontal_flip=True,
            fill_mode="nearest")
        for img_name in list_of_images:
            tmp_img_name = os.path.join(data_dir, img_name)
            img = load_img(tmp_img_name)
            img = img_to_array(img)
            img = img.reshape((1,) + img.shape)

            batch = datagen.flow(img,
                batch_size=1,
                seed=21,
                save_to_dir=data_dir,
                save_prefix=img_name.split(".JPG")[0] + "augmented",
                save_format="JPG")

            batch.next()
```

Figure 16: Oversampling minority class

```
classes_to_augment = [
    "Red_Squirrel",
    "European_Hare",
    "Ocelot",
    "Collared_Peccary",
    "White-nosed_Coati"]

for class_names in classes_to_augment:
    print("Currently Augmenting:", class_names)
    data_dir = os.path.join(train_folder, class_names)
    data_augment(data_dir)
```

Figure 17: Classes to augment

```
Currently Augmenting: Red_Squirrel
Currently Augmenting: European_Hare
Currently Augmenting: Ocelot
Currently Augmenting: Collared_Peccary
Currently Augmenting: White-nosed_Coati
```

Figure 18: Classes augmenting

Hansson (2002) augmented images before calculating top-1 and top-5 values.

2.7 Resizing of images

```
[ ] from pydrive.auth import GoogleAuth
    from pydrive.drive import GoogleDrive
    from google.colab import auth
    from oauth2client.client import GoogleCredentials

[ ] auth.authenticate_user()
    gauth = GoogleAuth()

    WARNING:tensorflow:
    The TensorFlow contrib module will not be included in TensorFlow 2.0.
    For more information, please see:
      * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
      * https://github.com/tensorflow/addons
      * https://github.com/tensorflow/io (for I/O related ops)
    If you depend on functionality not listed there, please file an issue.

[ ] gauth.credentials = GoogleCredentials.get_application_default()
    drive = GoogleDrive(gauth)

[ ] your_module = drive.CreateFile({"id": "1SLIjmWvYhFEQ6ImUlOzv5rZa4eV35eE5"})   # "your_module_file_id" is the part after "id=" in the shareable link
    your_module.GetContentFile("six_classes_utils.py") # Save the .py module file to Colab VM
```

Figure 19: File access on google drive

```
[ ] import six_classes_utils

[ ] from multiprocessing import Pool
```

Figure 20: Importing the functions from an external file

Files from google drive are accessed by setting up the gauth function. Shareable link is used to import functions from a different file. The external file six_classes_utils includes resizing function for the images.

```
[ ] from functools import partial

    #Resize Images
    if __name__ == '__main__':
        pool = Pool()
        image_list = glob.glob(train_folder + "/*/*")
        func = partial(six_classes_utils.resize_image, size=299)
        pool.map(func, image_list)
        pool.close()

    six_classes_utils.display_images(train_folder)
```

Figure 21: Resize of the images

six_classes_utils is the file used from google drive to resie the images. Libraries like cv2, random, glob are used and a function for resizing is written in the file.[3]

[3]https://opencv.org/

```
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
from PIL import Image
import cv2
import os
```

Figure 22: Libraries for the util file

```
def resize_image(file, size=299):
    img = Image.open(file)
    img = img.resize((size,size))
    img.save(file)
```

Figure 23: Function for resizing

Figure 24: Sample image after resizing

2.8 Look at Distribution of Selected Classes again

Now, the class imbalance issue has been resolved as all the classes are more likely similar in the number of images that each class possess. Pygal barchart is used for plotting the distribution graph.

```
#Compare class distribution
line_chart = pygal.Bar(height=300)
line_chart.title = 'Animals Class Distribution'
for o in os.listdir(train_folder):
    line_chart.add(o, len(os.listdir(os.path.join(train_folder, o))))
galplot(line_chart)
```

Figure 25: Class Distribution after augmentation

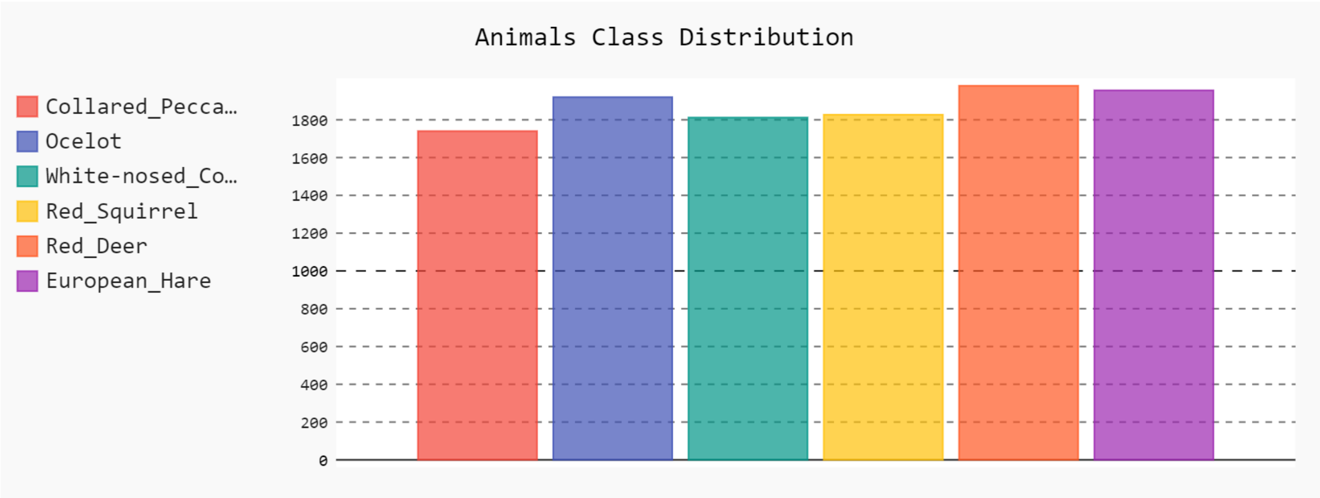

Figure 26: Bar chart after augmentation

3 Deep Learning Architectures

3.1 Data Generator

```
[ ] from keras.preprocessing.image import ImageDataGenerator

Using TensorFlow backend.

[ ] from keras.applications.inception_v3 import preprocess_input, decode_predictions

[ ] #Mount the google drive:
    from google.colab import drive
    drive.mount('/content/drive')

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect

Enter your authorization code:
··········
Mounted at /content/drive

[ ] #Check the directory:
    cd /content/drive/My Drive/Colab Notebooks/Six_Classes

/content/drive/My Drive/Colab Notebooks/Six_Classes
```

Figure 27: Importing ImageGenerator

```
[ ] WIDTH=299
    HEIGHT=299
    BATCH_SIZE=64
    test_dir = 'test/'
    train_dir = 'train/'
    val_dir = 'val/'
```

Figure 28: Resizing as per InceptionV3

```
[ ] WIDTH=224
    HEIGHT=224
    BATCH_SIZE=64
    test_dir = 'test/'
    train_dir = 'train/'
    val_dir = 'val/'
```

Figure 29: Resizing as per VGG16 and MobileNet

Nguyen et al. (2018) resized the images before applying the deep learning architectures.

```
[ ] #Train DataSet Generator with Augmentation
    print("\nTraining Data Set")
    train_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
    train_flow = train_generator.flow_from_directory(
        train_dir,
        target_size=(HEIGHT, WIDTH),
        batch_size = BATCH_SIZE
    )

    Training Data Set
    Found 11209 images belonging to 6 classes.
```

Figure 30: Train Dataset Generator

Verma and Gupta (2018) applied DCNN architectures after training and testing dataset using generator.

```
[ ] #Validation DataSet Generator with Augmentation
    print("\nValidation Data Set")
    val_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
    val_flow = val_generator.flow_from_directory(
        val_dir,
        target_size=(HEIGHT, WIDTH),
        batch_size = BATCH_SIZE
    )

    Validation Data Set
    Found 622 images belonging to 6 classes.
```

Figure 31: Validation Dataset Generator

```
[ ] #Test DataSet Generator with Augmentation
    print("\nTest Data Set")
    test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
    test_flow = test_generator.flow_from_directory(
            test_dir,
            target_size=(HEIGHT, WIDTH),
            batch_size = BATCH_SIZE
    )
```

```
Test Data Set
Found 1256 images belonging to 6 classes.
```

Figure 32: Test Dataset Generator

The data from the Keras ImageDataGenerator class has been ingested for training purposes.[4] This will assist in reading the directory structured as per the category of classes which was done during the training in the exploration phase.

For InceptionV3, the height and width requirement is 299x299. For VGG-16 and MobileNet, the height and width requirement is 224x224. The generator also resizes the images as per the architecture before feeding the data into the network so that the training, test and validation phase works successfully. Chen et al. (2014) and Chung et al. (2018) resized the images before applying the DCNN algorithms.

3.2 Optimization for CPU

```
[ ] from keras.models import Sequential, Model, load_model
    from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, CSVLogger
    from keras import optimizers, models
    from keras.layers import Dense, Dropout, GlobalAveragePooling2D
    from keras import applications
    from keras import backend as K
    import tensorflow as tf
    import os
```

Figure 33: Libraries for optimizers and modelst

[4]https://keras.io/

```
[ ] NUM_PARALLEL_EXEC_UNITS = 8
```

```
[ ] #Set Performance Parameters for MKL and Tensorflow using Keras backend
    #TensorFlow
    config = tf.ConfigProto(
            intra_op_parallelism_threads=NUM_PARALLEL_EXEC_UNITS,
            inter_op_parallelism_threads=1
    )

    session = tf.Session(config=config)
    K.set_session(session)
```

```
[ ] #MKL and OpenMP
    os.environ["OMP_NUM_THREADS"] = str(NUM_PARALLEL_EXEC_UNITS)
    os.environ["KMP_BLOCKTIME"] = "1"
    os.environ["KMP_SETTINGS"] = "1"
    os.environ["KMP_AFFINITY"]= "granularity=fine,verbose,compact,1,0"
```

Figure 34: Optimization setup

3.3 Selecting Hyperparameters

```
[ ] # Initialize InceptionV3 with transfer learning
    base_model = applications.InceptionV3(weights='imagenet',
                                          include_top=False,
                                          input_shape=(WIDTH, HEIGHT,3))
```

Figure 35: Model for InceptionV3

```
[ ] # add a global spatial average pooling layer
    x = base_model.output
```

```
[ ] x = GlobalAveragePooling2D()(x)
    # and a dense layer
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(len(train_flow.class_indices), activation='softmax')(x)
```

```
[ ] # this is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)
```

```
[ ] # first: train only the top layers (which were randomly initialized)
    # i.e. freeze all convolutional InceptionV3 layers
    for layer in base_model.layers:
        layer.trainable = False
```

```
[ ] # compile the model (should be done *after* setting layers to non-trainable)
    model.compile(optimizer=optimizers.Adam(lr=0.001), metrics=['accuracy', 'top_k_categorical_accuracy'], loss='categorical_crossentropy')
    model.summary()
```

Figure 36: Layers for InceptionV3

```
Model: "model_1"
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 299, 299, 3)  0
____________________________________________________________________________________________________
conv2d_1 (Conv2D)                (None, 149, 149, 32) 864         input_1[0][0]
____________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 149, 149, 32) 96          conv2d_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 149, 149, 32) 0           batch_normalization_1[0][0]
____________________________________________________________________________________________________
conv2d_2 (Conv2D)                (None, 147, 147, 32) 9216        activation_1[0][0]
____________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 147, 147, 32) 96          conv2d_2[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 147, 147, 32) 0           batch_normalization_2[0][0]
____________________________________________________________________________________________________
conv2d_3 (Conv2D)                (None, 147, 147, 64) 18432       activation_2[0][0]
____________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 147, 147, 64) 192         conv2d_3[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 147, 147, 64) 0           batch_normalization_3[0][0]
____________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)   (None, 73, 73, 64)   0           activation_3[0][0]
____________________________________________________________________________________________________
conv2d_4 (Conv2D)                (None, 73, 73, 80)   5120        max_pooling2d_1[0][0]
____________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 73, 73, 80)   240         conv2d_4[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 73, 73, 80)   0           batch_normalization_4[0][0]
____________________________________________________________________________________________________
conv2d_5 (Conv2D)                (None, 71, 71, 192)  138240      activation_4[0][0]
```

Figure 37: Model summary for InceptionV3 (1)

```
____________________________________________________________________________________________________
activation_93 (Activation)       (None, 8, 8, 384)    0           batch_normalization_93[0][0]
____________________________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 8, 8, 192)    576         conv2d_94[0][0]
____________________________________________________________________________________________________
activation_86 (Activation)       (None, 8, 8, 320)    0           batch_normalization_86[0][0]
____________________________________________________________________________________________________
mixed9_1 (Concatenate)           (None, 8, 8, 768)    0           activation_88[0][0]
                                                                  activation_89[0][0]
____________________________________________________________________________________________________
concatenate_2 (Concatenate)      (None, 8, 8, 768)    0           activation_92[0][0]
                                                                  activation_93[0][0]
____________________________________________________________________________________________________
activation_94 (Activation)       (None, 8, 8, 192)    0           batch_normalization_94[0][0]
____________________________________________________________________________________________________
mixed10 (Concatenate)            (None, 8, 8, 2048)   0           activation_86[0][0]
                                                                  mixed9_1[0][0]
                                                                  concatenate_2[0][0]
                                                                  activation_94[0][0]
____________________________________________________________________________________________________
global_average_pooling2d_3 (Glo (None, 2048)         0           mixed10[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1024)         2098176     global_average_pooling2d_3[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 6)            6150        dense_2[0][0]
====================================================================================================
Total params: 23,907,110
Trainable params: 2,104,326
Non-trainable params: 21,802,784
```

Figure 38: Model summary for InceptionV3 (2)

```
[ ]  # Initialize mobilenet with transfer learning
     base_model = applications.MobileNet(weights='imagenet',
                                         include_top=False,
                                         input_shape=(WIDTH, HEIGHT,3))
```

Figure 39: Model summary for MobileNet (1)

```
_________________________________________________________________
conv_pw_12_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_dw_13 (DepthwiseConv2D) (None, 7, 7, 1024)        9216
_________________________________________________________________
conv_dw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576
_________________________________________________________________
conv_pw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 6150
=================================================================
Total params: 4,284,614
Trainable params: 1,055,750
Non-trainable params: 3,228,864
```

Figure 40: Model summary for MobileNet (2)

```
[ ]  # Initialize VGG16 with transfer learning
     base_model = applications.VGG16(weights='imagenet',
                                     include_top=False,
                                     input_shape=(WIDTH, HEIGHT,3))
```

Figure 41: Model summary for VGG16 (1)

```
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              525312
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 6150
=================================================================
Total params: 15,246,150
Trainable params: 531,462
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 42: Model summary for VGG16 (2)

InceptionV3, VGG16 and MobileNet are the three architectures used for the classification of animals. Here, transfer learning is used with the weights of imagenet and top layer is removed as 1001 classes are not predicted in this dataset. The learning rate is set to 0.001 and therefore, the dataset takes a longer time to train for each of the algorithms. Adam optimizer is used as it adapts to the learning rate according to the parameters. Batch size of 32 is used and hence, 32 images are used in training the dataset in one iteration.

GlobalAveragePooling2D layer is added to the base model and the Dense layer is added with a softmax activation to predict the number of classes in the dataset. Layer.trainables is set to False so that the new layers are trained that are added in this dataset. Since, we have multiclass classification, we have added loss of categorical crossentropy is used.

3.4 Training Callbacks

Keras Fit Generator Method is used for training. Here, training and validation dataset is used in order to check if the model is performing well rather than directly working on the test dataset. Four different callbacks such as ModelCheckkpoint, TensorBoard, EarlyStopping and CSVLogger. Checkpoints are used to minimize the disk space that is being used. Also, overtraining and overutilizing the compute is taken care of by callbacks.

```
[ ] import math
    top_layers_file_path="top_layers.iv3.hdf5"
```

```
[ ] checkpoint = ModelCheckpoint(top_layers_file_path, monitor='loss', verbose=1, save_best_only=True, mode='min')
    tb = TensorBoard(log_dir='./logs', batch_size=val_flow.batch_size, write_graph=True, update_freq='batch')
    early = EarlyStopping(monitor="loss", mode="min", patience=5)
    csv_logger = CSVLogger('./logs/iv3-log.csv', append=True)
```

Figure 43: Model checkpoint for InceptionV3

```
[ ] history = model.fit_generator(train_flow,
                                  epochs=3,
                                  verbose=1,
                                  validation_data=val_flow,
                                  validation_steps=math.ceil(val_flow.samples/val_flow.batch_size),
                                  steps_per_epoch=math.ceil(train_flow.samples/train_flow.batch_size),
                                  callbacks=[checkpoint, early, tb, csv_logger])
```

Figure 44: Training for InceptionV3

```
Epoch 1/3
  1/176 [..............................] - ETA: 2:18:17 - loss: 1.8698 - acc: 0.1406 - top_k_categorical_accuracy: 0.7812WARNING:tensorflow:From /usr/local/lib/p

176/176 [==============================] - 3637s 21s/step - loss: 0.6026 - acc: 0.7977 - top_k_categorical_accuracy: 0.9909 - val_loss: 1.0850 - val_acc: 0.6929

Epoch 00001: loss improved from inf to 0.60256, saving model to top_layers.iv3.hdf5
Epoch 2/3
176/176 [==============================] - 3661s 21s/step - loss: 0.2026 - acc: 0.9321 - top_k_categorical_accuracy: 0.9999 - val_loss: 1.4542 - val_acc: 0.6913

Epoch 00002: loss improved from 0.60256 to 0.20270, saving model to top_layers.iv3.hdf5
Epoch 3/3
176/176 [==============================] - 3426s 19s/step - loss: 0.1347 - acc: 0.9563 - top_k_categorical_accuracy: 1.0000 - val_loss: 1.6536 - val_acc: 0.6849

Epoch 00003: loss improved from 0.20270 to 0.13483, saving model to top_layers.iv3.hdf5
```

Figure 45: Output for training and validation of InceptionV3

```
[ ] import math
    top_layers_file_path="top_layers.mn.hdf5"
```

Figure 46: Model checkpoint for MobileNet (1)

```
[ ] checkpoint = ModelCheckpoint(top_layers_file_path, monitor='loss', verbose=1, save_best_only=True, mode='min')
    tb = TensorBoard(log_dir='./logs', batch_size=val_flow.batch_size, write_graph=True, update_freq='batch')
    early = EarlyStopping(monitor="loss", mode="min", patience=5)
    csv_logger = CSVLogger('./logs/mn-log.csv', append=True)
```

Figure 47: Model checkpoint for MobileNet (2)

```
[ ] history = model.fit_generator(train_flow,
                                  epochs=3,
                                  verbose=1,
                                  validation_data=val_flow,
                                  validation_steps=math.ceil(val_flow.samples/val_flow.batch_size),
                                  steps_per_epoch=math.ceil(train_flow.samples/train_flow.batch_size),
                                  callbacks=[checkpoint, early, tb, csv_logger])
```

Figure 48: Training for MobileNet

```
Epoch 1/3
176/176 [==============================] - 65s 369ms/step - loss: 0.0518 - acc: 0.9843 - top_k_categorical_accuracy: 1.0000 - val_loss: 1.1926 - val_acc: 0.5965

Epoch 00001: loss improved from 0.33142 to 0.05163, saving model to top_layers.mn.hdf5
Epoch 2/3
176/176 [==============================] - 64s 362ms/step - loss: 0.0496 - acc: 0.9838 - top_k_categorical_accuracy: 0.9999 - val_loss: 0.9246 - val_acc: 0.7186

Epoch 00002: loss improved from 0.05163 to 0.04967, saving model to top_layers.mn.hdf5
Epoch 3/3
176/176 [==============================] - 63s 357ms/step - loss: 0.0234 - acc: 0.9928 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.9338 - val_acc: 0.7251

Epoch 00003: loss improved from 0.04967 to 0.02343, saving model to top_layers.mn.hdf5
```

Figure 49: Output for training and validation of MobileNet

```
[ ] import math
    top_layers_file_path="top_layers.vgg16.hdf5"
```

Figure 50: Model checkpoint for VGG-16 (1)

```
[ ] checkpoint = ModelCheckpoint(top_layers_file_path, monitor='loss', verbose=1, save_best_only=True, mode='min')
    tb = TensorBoard(log_dir='./logs', batch_size=val_flow.batch_size, write_graph=True, update_freq='batch')
    early = EarlyStopping(monitor="loss", mode="min", patience=5)
    csv_logger = CSVLogger('./logs/vgg16-log.csv', append=True)
```

Figure 51: Model checkpoint for VGG-16 (2)

```
[ ] history = model.fit_generator(train_flow,
                                  epochs=3,
                                  verbose=1,
                                  validation_data=val_flow,
                                  validation_steps=math.ceil(val_flow.samples/val_flow.batch_size),
                                  steps_per_epoch=math.ceil(train_flow.samples/train_flow.batch_size),
                                  callbacks=[checkpoint, early, tb, csv_logger])
```

Figure 52: Training for VGG-16

```
Epoch 1/3
  1/176 [..............................] - ETA: 2:39:42 - loss: 4.8683 - acc: 0.1562 - top_k_categorical_accuracy: 0.8281WARNING:tensorflow:From /usr/local/lib/p

176/176 [==============================] - 6063s 34s/step - loss: 0.4321 - acc: 0.8970 - top_k_categorical_accuracy: 0.9899 - val_loss: 1.9332 - val_acc: 0.5193

Epoch 00001: loss improved from inf to 0.43310, saving model to top_layers.vgg16.hdf5
Epoch 2/3
176/176 [==============================] - 6044s 34s/step - loss: 0.0513 - acc: 0.9847 - top_k_categorical_accuracy: 1.0000 - val_loss: 2.4323 - val_acc: 0.5048

Epoch 00002: loss improved from 0.43310 to 0.05128, saving model to top_layers.vgg16.hdf5
Epoch 3/3
176/176 [==============================] - 6051s 34s/step - loss: 0.0399 - acc: 0.9875 - top_k_categorical_accuracy: 1.0000 - val_loss: 2.5020 - val_acc: 0.5161

Epoch 00003: loss improved from 0.05128 to 0.04000, saving model to top_layers.vgg16.hdf5
```

Figure 53: Output for training and validation of VGG-16

3.5 Evaluate Model

```
[ ] model.load_weights(top_layers_file_path)
    loss, acc, top_5 = model.evaluate_generator(
        test_flow,
        verbose = True,
        steps=math.ceil(test_flow.samples/test_flow.batch_size))
```

Figure 54: Evaluation generator for InceptionV3

```
[ ] print("Loss: ", loss)

Loss:  1.6115237535185116

[ ] print("Acc: ", acc)

Acc:  0.695859872611465

[ ] print("Top 5: ", top_5)

Top 5:  0.9976114649681529
```

Figure 55: Loss, accuracy and Top5 confidence for InceptionV3

```
[ ] model.load_weights(top_layers_file_path)
    loss, acc, top_5 = model.evaluate_generator(
        test_flow,
        verbose = True,
        steps=math.ceil(test_flow.samples/test_flow.batch_size))
    print("Loss: ", loss)
    print("Acc: ", acc)
    print("Top 5: ", top_5)

    20/20 [==============================] - 492s 25s/step
    Loss:  1.3548380257977042
    Acc:  0.6791401277681824
    Top 5:  0.9976114657274477
```

Figure 56: Loss, accuracy and Top5 confidence for MobileNet

```
[ ] model.load_weights(top_layers_file_path)
    loss, acc, top_5 = model.evaluate_generator(
        test_flow,
        verbose = True,
        steps=math.ceil(test_flow.samples/test_flow.batch_size))
    print("Loss: ", loss)
    print("Acc: ", acc)
    print("Top 5: ", top_5)

    20/20 [==============================] - 835s 42s/step
    Loss:  1.8602820445018209
    Acc:  0.6242038220356984
    Top 5:  0.9992038216560509
```

Figure 57: Loss, accuracy and Top5 confidence for VGG-16

3.6 Write Labels File

```
[ ] label = [k for k,v in train_flow.class_indices.items()]
```

```
[ ] with open('iv3-labels.txt', 'w+') as file:
        file.write("\n".join(label))
```

Figure 58: Write Label for IV3

```
[ ] label = [k for k,v in train_flow.class_indices.items()]
    with open('mn-labels.txt', 'w+') as file:
        file.write("\n".join(label))
```

Figure 59: Write Label for MobileNet

```
[ ] label = [k for k,v in train_flow.class_indices.items()]
    with open('vgg16-labels.txt', 'w+') as file:
        file.write("\n".join(label))
```

Figure 60: Write Label for VGG-16

The network uses the numerical value to refer to a particular class. The values are saved in a text file to map the classes to a particular numerical value for future use.

3.7 Test Model with Sample Image

```
[ ]  from keras.preprocessing import image
     import numpy as np
     import glob
     import random
```

Figure 61: Test Model for InceptionV3 (1)

```
[ ]  file_list = glob.glob("test/*/*")

[ ]  img_path = random.choice(file_list)

[ ]  img_cat = os.path.split(os.path.dirname(img_path))[1]

[ ]  print("Image Category: ", img_cat)

     Image Category:  White-nosed_Coati
```

Figure 62: Test Model for InceptionV3 (2)

```
[ ]  img = image.load_img(img_path, target_size=(299, 299))

[ ]  x = image.img_to_array(img)

[ ]  x = np.expand_dims(x, axis=0)

[ ]  x = preprocess_input(x)

[ ]  preds = model.predict(x)

[ ]  print("Raw Predictions: ", preds)

     Raw Predictions:  [[9.9079716e-01 1.3424299e-10 5.9693298e-06 2.2137892e-06 9.6429358e-05
       9.0981722e-03]]
```

Figure 63: Test Model for InceptionV3 (3)

```
[ ] top_x = 3
```

```
[ ] top_args = preds[0].argsort()[-top_x:][::-1]
```

```
[ ] preds_label = [label[p] for p in top_args]
```

Figure 64: Test Model for InceptionV3 (4)

```
[ ] print("\nTop " + str(top_x) + " confidence: " + " ".join(map(str, sorted(preds[0])[-top_x:][::-1])))

    Top 3 confidence: 0.99079716 0.009098172 9.642936e-05
```

```
[ ] print("Top " + str(top_x) + " labels: " + " ".join(map(str, preds_label)))

    Top 3 labels: Collared_Peccary White-nosed_Coati Red_Squirrel
```

Figure 65: Test Model for InceptionV3 (5)

```
[ ] from keras.preprocessing import image
    import numpy as np
    import glob
    import random

    file_list = glob.glob("test/*/*")
    img_path = random.choice(file_list)
    img_cat = os.path.split(os.path.dirname(img_path))[1]
    print("Image Category: ", img_cat)
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    preds = model.predict(x)
    print("Raw Predictions: ", preds)

    top_x = 3
    top_args = preds[0].argsort()[-top_x:][::-1]
    preds_label = [label[p] for p in top_args]
    print("\nTop " + str(top_x) + " confidence: " + " ".join(map(str, sorted(preds[0])[-top_x:][::-1])))
    print("Top " + str(top_x) + " labels: " + " ".join(map(str, preds_label)))
```

Figure 66: Test Model for MobileNet (1)

```
Image Category:  Red_Deer
Raw Predictions:  [[6.9340802e-04 1.1363633e-05 1.0804304e-06 9.9020493e-01 5.6083937e-08
  9.0891048e-03]]

Top 3 confidence: 0.99020493 0.009089105 0.000693408
Top 3 labels: Red_Deer White-nosed_Coati Collared_Peccary
```

Figure 67: Test Model for MobileNet (2)

```
[ ]  from keras.preprocessing import image
     import numpy as np
     import glob
     import random
```

```
[ ]  file_list = glob.glob("test/*/*")
     img_path = random.choice(file_list)
     img_cat = os.path.split(os.path.dirname(img_path))[1]
     print("Image Category: ", img_cat)
     img = image.load_img(img_path, target_size=(224, 224))
     x = image.img_to_array(img)
     x = np.expand_dims(x, axis=0)
     x = preprocess_input(x)
```

```
Image Category:  White-nosed_Coati
```

Figure 68: Test Model for VGG-16 (1)

```
[ ]  preds = model.predict(x)
     print("Raw Predictions: ", preds)
```

```
Raw Predictions:  [[9.3152217e-04 5.0856639e-04 1.2653039e-03 5.9000980e-05 8.0474538e-01
  1.9249023e-01]]
```

Figure 69: Test Model for VGG-16 (2)

```
[ ] top_x = 3
    top_args = preds[0].argsort()[-top_x:][::-1]
    preds_label = [label[p] for p in top_args]
    print("\nTop " + str(top_x) + " confidence: " + " ".join(map(str, sorted(preds[0])[-top_x:][::-1])))
    print("Top " + str(top_x) + " labels: " + " ".join(map(str, preds_label)))

    Top 3 confidence: 0.8047454 0.19249023 0.0012653039
    Top 3 labels: Red_Squirrel White-nosed_Coati Ocelot
```

Figure 70: Test Model for VGG-16 (3)

The models are first tested on a sample image after the training. A random image is chosen from the test dataset and the model is run through the image. Softmax function returns the confidence values. Top-3 labels and values are predicted by the model. Norouzzadeh et al. (2017) calculated and Gomez Villa et al. (2017) discussed about the confidence values to save human labor.

3.8 Transform Keras Model to Tensorflow

```
[ ] from tensorflow.python.framework import graph_util
    from tensorflow.python.framework import graph_io
```

Figure 71: Tranformation from Keras to Tensorflow for InceptionV3 (1)

```
[ ] input_model_path = top_layers_file_path

[ ] output_model_name = "top_layers.iv3.pb"

[ ] output_model_dir = "tf_model"

[ ] K.set_learning_phase(0)
    sess = K.get_session()

[ ] test_model = models.load_model(input_model_path)
    orig_output_node_names = [node.op.name for node in test_model.outputs]

[ ] constant_graph = graph_util.convert_variables_to_constants(
        sess,
        sess.graph.as_graph_def(),
        orig_output_node_names)
```

Figure 72: Tranformation from Keras to Tensorflow for InceptionV3 (2)

```
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
INFO:tensorflow:Froze 380 variables.
INFO:tensorflow:Converted 380 variables to const ops.
```

Figure 73: Tranformation from Keras to Tensorflow for InceptionV3 (3)

```
[ ] graph_io.write_graph(
        constant_graph,
        output_model_dir,
        output_model_name,
        as_text=False)

'tf_model/top_layers.iv3.pb'
```

Figure 74: Tranformation from Keras to Tensorflow for InceptionV3 (4)

```
from tensorflow.python.framework import graph_util
from tensorflow.python.framework import graph_io

input_model_path = top_layers_file_path
output_model_name = "top_nodes.mn.pb"
output_model_dir = "tf_model"

K.set_learning_phase(0)
sess = K.get_session()

test_model = models.load_model(input_model_path)
orig_output_node_names = [node.op.name for node in test_model.outputs]

constant_graph = graph_util.convert_variables_to_constants(
    sess,
    sess.graph.as_graph_def(),
    orig_output_node_names)
graph_io.write_graph(
    constant_graph,
    output_model_dir,
    output_model_name,
    as_text=False)
```

Figure 75: Tranformation from Keras to Tensorflow for MobileNet (1)

```
Use `tf.compat.v1.graph_util.extract_sub_graph`
INFO:tensorflow:Froze 139 variables.
INFO:tensorflow:Converted 139 variables to const ops.
'tf_model/top_nodes.mn.pb'
```

Figure 76: Tranformation from Keras to Tensorflow for MobileNet (2)

```
[ ] input_model_path = top_layers_file_path
    output_model_name = "top_layers.vgg16.pb"
    output_model_dir = "tf_model"
```

```
[ ] K.set_learning_phase(0)
    sess = K.get_session()
```

```
[ ] test_model = models.load_model(input_model_path)
    orig_output_node_names = [node.op.name for node in test_model.outputs]
```

Figure 77: Tranformation from Keras to Tensorflow for VGG-16 (1)

```
[ ] constant_graph = graph_util.convert_variables_to_constants(
            sess,
            sess.graph.as_graph_def(),
            orig_output_node_names)
    graph_io.write_graph(
            constant_graph,
            output_model_dir,
            output_model_name,
            as_text=False)
```

Figure 78: Tranformation from Keras to Tensorflow for VGG-16 (2)

```
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
INFO:tensorflow:Froze 30 variables.
INFO:tensorflow:Converted 30 variables to const ops.
'tf_model/top_layers.vgg16.pb'
```

Figure 79: Tranformation from Keras to Tensorflow for VGG-16 (3)

This step is followed to convert the .hdf5 file format of Keras to .pb file format of TensorFlow. The files for all the three architectures are saved so that it can be used in the future if a researcher wants to use TensorFlow instead of Keras.

4 Model Analysis

4.1 Loading the models for evaluation

```
[ ] from keras.models import load_model
    model = load_model('top_layers.iv3.hdf5')

[ ] from keras.preprocessing.image import ImageDataGenerator
    from keras.applications.inception_v3 import preprocess_input
```

Figure 80: Load model of InceptionV3 for evaluation

```
[ ] #Test DataSet Generator with Augmentation
    test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)

[ ] test_flow = test_generator.flow_from_directory(
        'test',
        shuffle=False,
        target_size=(299, 299),
        batch_size = 32
    )

    Found 1256 images belonging to 6 classes.
```

Figure 81: Test dataset generator of InceptionV3 for evaluation

```
[ ] import math
    import numpy as np
```

Figure 82: Importing libraries for InceptionV3 for evaluation

```
[ ] predictions = model.predict_generator(
        test_flow,
        verbose=1,
        steps=math.ceil(test_flow.samples/test_flow.batch_size))
    predicted_classes = np.argmax(predictions, axis=1)

    40/40 [==============================] - 320s 8s/step
```

Figure 83: Predict generator of InceptionV3 for evaluation (1)

```
[ ] true_classes = test_flow.classes
```

```
[ ] class_labels = list(test_flow.class_indices.keys())
```

Figure 84: Predict generator of InceptionV3 for evaluation (2)

```
[ ] from keras.models import load_model
    model = load_model('top_layers.mn.hdf5')
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    from keras.applications.mobilenet import preprocess_input
```

```
[ ] #Test DataSet Generator with Augmentation
    test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
    test_flow = test_generator.flow_from_directory(
        'test',
        shuffle=False,
        target_size=(224, 224),
        batch_size = 32
    )

    Found 1256 images belonging to 6 classes.
```

Figure 85: Load model of MobileNet for evaluation

```
[ ] from keras.models import load_model
    model = load_model('top_layers.vgg16.hdf5')
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    from keras.applications import vgg16
    #Test DataSet Generator with Augmentation
    test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
    test_flow = test_generator.flow_from_directory(
        'test',
        shuffle=False,
        target_size=(224, 224),
        batch_size = 32
    )

    Found 1256 images belonging to 6 classes.
```

Figure 86: Load model of VGG-16 for evaluation

Models for all the three architectures, InceptionV3, MobileNet and VGG-16 are loaded. Predict generator is used to generate predictions and passed onto the dataset which is further used by the analysis functions. Batch size is set to 32 and Target size of the images is 299x299 for InceptionV3 and 224x224 for MobileNet and VGG-16.

4.2 Confusion Matrix

```
[ ] import matplotlib.pyplot as plt
    %matplotlib inline
    import scikitplot as skplt
```

Figure 87: Code for the confusion matrix (1)

```
[ ] [print(k, ":", v) for k,v in enumerate(class_labels)]

    0 : Collared_Peccary
    1 : European_Hare
    2 : Ocelot
    3 : Red_Deer
    4 : Red_Squirrel
    5 : White-nosed_Coati
```

Figure 88: Code for the confusion matrix (2)

```
[ ] true_map_classes = [class_labels[x] for x in true_classes]
```

```
[ ] predicted_map_classes = [class_labels[x] for x in predicted_classes]
```

Figure 89: Code for the confusion matrix (3)

```
skplt.metrics.plot_confusion_matrix(
    true_map_classes,
    predicted_map_classes,
    labels=class_labels,
    x_tick_rotation=90,
    figsize=(12,12))
```

Figure 90: Code for the confusion matrix (4)

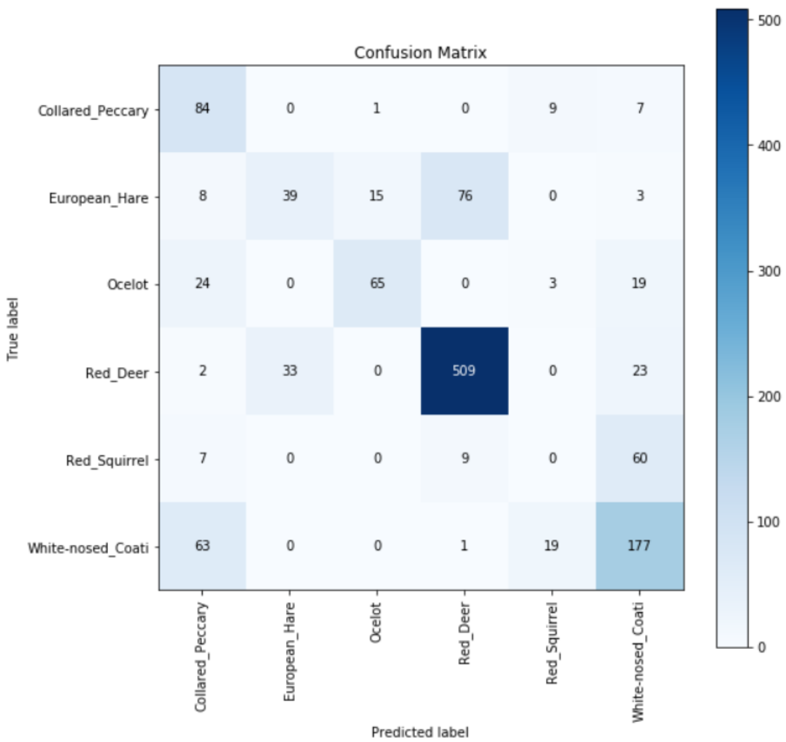

Figure 91: Confusion Matrix for InceptionV3

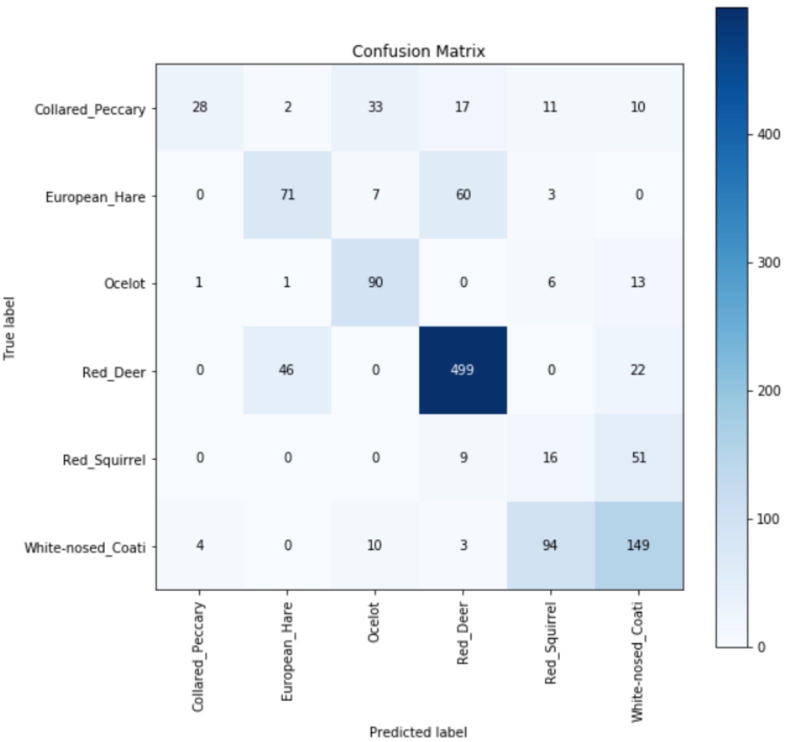

Figure 92: Confusion Matrix for MobileNet

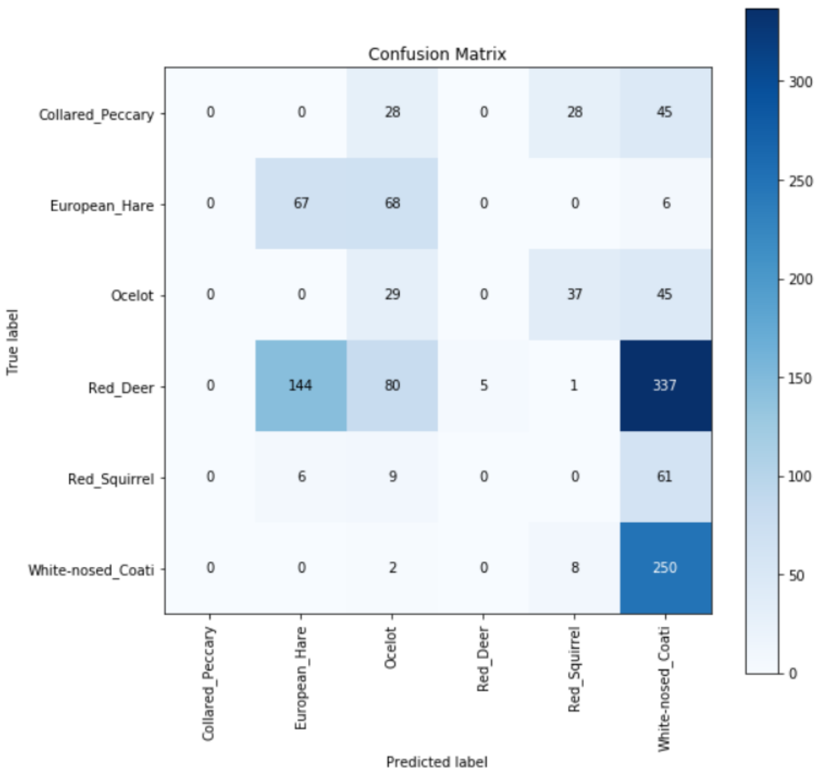

Figure 93: Confusion Matrix for VGG-16

4.3 Classification Report

```
[ ] from sklearn.metrics import classification_report
```

```
[ ] report = classification_report(
        true_classes,
        predicted_classes,
        target_names=class_labels)
```

```
[ ] print(report)
```

Figure 94: Code for the classification report

```
                   precision    recall  f1-score   support

  Collared_Peccary       0.45      0.83      0.58       101
     European_Hare       0.54      0.28      0.37       141
            Ocelot       0.80      0.59      0.68       111
          Red_Deer       0.86      0.90      0.88       567
      Red_Squirrel       0.00      0.00      0.00        76
White-nosed_Coati       0.61      0.68      0.64       260

          accuracy                           0.70      1256
         macro avg       0.54      0.55      0.52      1256
      weighted avg       0.68      0.70      0.68      1256
```

Figure 95: Classification report for InceptionV3

```
                   precision    recall  f1-score   support

  Collared_Peccary       0.85      0.28      0.42       101
     European_Hare       0.59      0.50      0.54       141
            Ocelot       0.64      0.81      0.72       111
          Red_Deer       0.85      0.88      0.86       567
      Red_Squirrel       0.12      0.21      0.16        76
White-nosed_Coati       0.61      0.57      0.59       260

          accuracy                           0.68      1256
         macro avg       0.61      0.54      0.55      1256
      weighted avg       0.71      0.68      0.68      1256
```

Figure 96: Classification report for MobileNet

	precision	recall	f1-score	support
Collared_Peccary	0.67	0.32	0.43	101
European_Hare	0.46	0.96	0.62	141
Ocelot	0.61	0.72	0.66	111
Red_Deer	0.99	0.66	0.79	567
Red_Squirrel	0.05	0.07	0.05	76
White-nosed_Coati	0.53	0.60	0.57	260
accuracy			0.62	1256
macro avg	0.55	0.55	0.52	1256
weighted avg	0.72	0.62	0.64	1256

Figure 97: Classification report for VGG-16

Classification report shows the values of precision, recall, f-1 score and support for each of the classes. Also, it shows the overall f-1 score for each of the models.

4.4 Precision-Recall Curve

```
[ ] skplt.metrics.plot_precision_recall(
        true_classes,
        predictions,
        figsize=(12,12))
```

Figure 98: Code for plotting the precision-recall curve

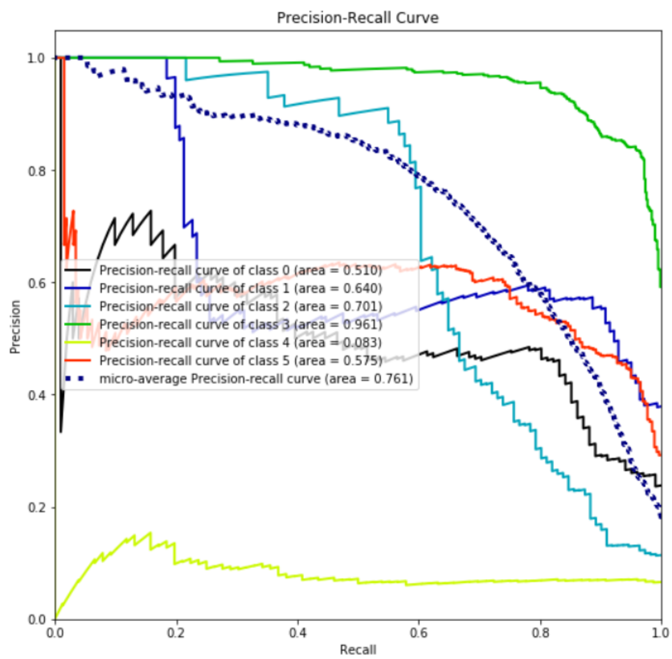

Figure 99: Precision-Recall Curve for InceptionV3

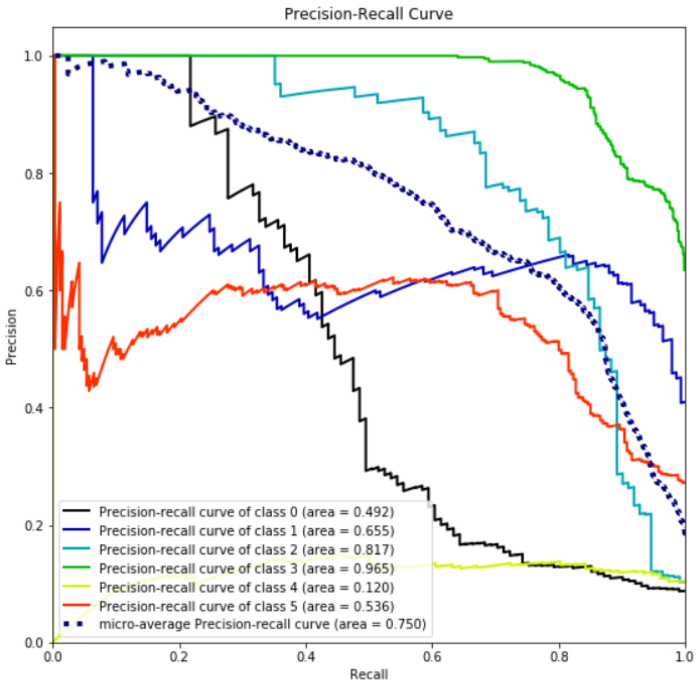

Figure 100: Precision-Recall Curve for MobileNet

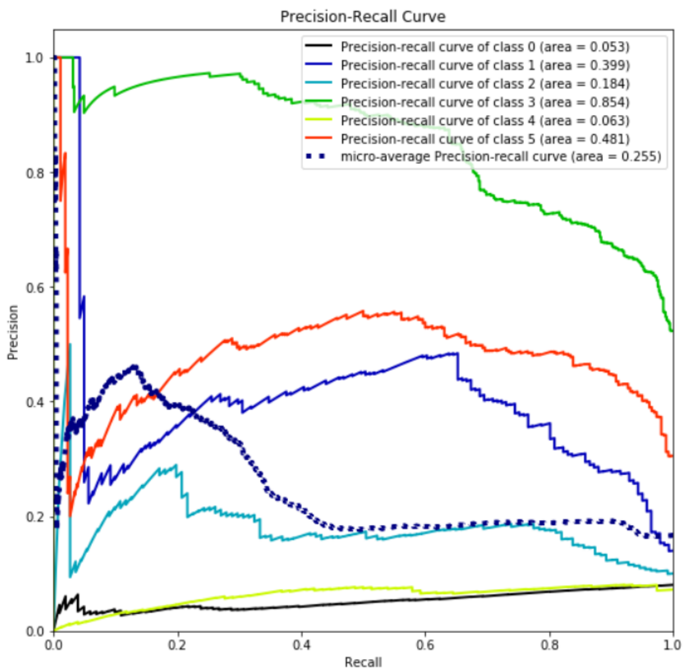

Figure 101: Precision-Recall Curve for VGG-16

Precision-Recall curve for the architectures InceptionV3, MobileNet and VGG-16 are plotted.

4.5 Receiver Operating Characteristic (ROC) Curve

A graphical plot which shows the binary classifier system as the threshold is varied. True positive rates is plotted against the fraction of False positive rates from the negatives at different threshold settings. True positive rate (TPR) is called as sensitivity, whereas False positive rate (FPR) is one minus the true negative rate.

```
skplt.metrics.plot_roc(
    true_classes,
    predictions,
    figsize=(12,12))
```

Figure 102: Code for plotting the ROC curve

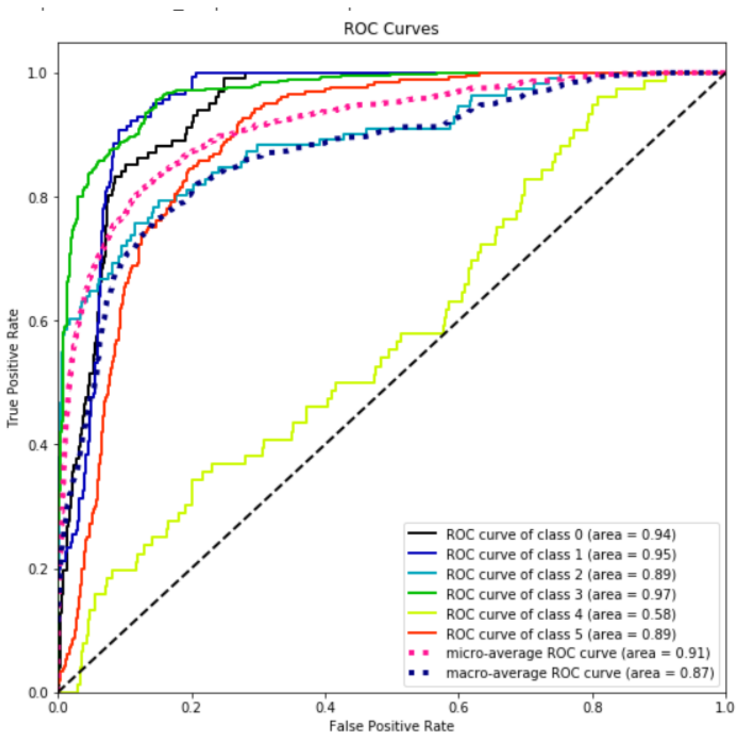

Figure 103: ROC Curve for InceptionV3

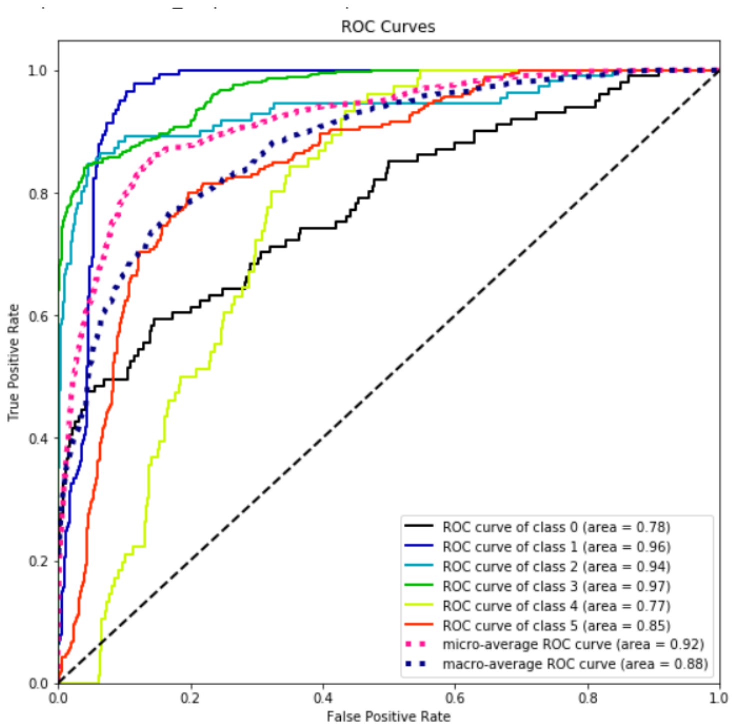

Figure 104: ROC Curve for MobileNet

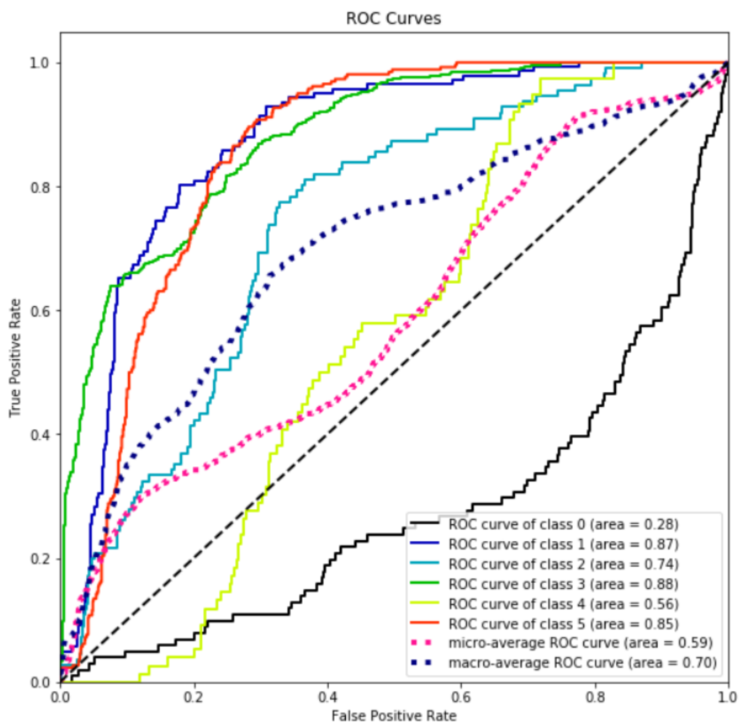

Figure 105: ROC Curve for VGG-16

ROC curves for InceptionV3, MobileNet and VGG-16 are plotted.

5 Environment Setup

Table 1: Hardware

Hardware Specification		
System RAM	Processor	Speed
8 GB	Intel CORE i5 8th Gen	1.80GHz Intel Core i5-8250U CPU and an AMD Radeon 530 GPU

Table 2: Software

Software Specification		
Google Colab, Python3, TensorFlow, Keras	OpenCV, NumPy	MatPlot Lib, ScikitPlot, Pygal

Figure 106: Environment Setup

References

Chen, G., Han, T. X., He, Z., Kays, R. and Forrester, T. (2014). Deep convolutional neural network based species recognition for wild animal monitoring, *2014 IEEE International Conference on Image Processing, ICIP 2014* pp. 858–862.

Chung, C., Patel, S., Lee, R., Fu, L., Reilly, S., Ho, T., Lionetti, J., George, M. D. and Taylor, P. (2018). Very Deep Convolutional Networks for Large-Scale Image Recognization, *[Vgg]* **75**(6): 398–406.

Gomez Villa, A., Salazar, A. and Vargas, F. (2017). Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks, *Ecological Informatics* **41**: 24–32.

Hansson, P. (2002). Fracture Analysis of Adhesive Joints Using The Finite Element Method, *Lund Institute of Technology* (February).

Nguyen, H., Maclagan, S. J., Nguyen, T. D., Nguyen, T., Flemons, P., Andrews, K., Ritchie, E. G. and Phung, D. (2018). Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring, *Proceedings - 2017 International Conference on Data Science and Advanced Analytics, DSAA 2017* **2018-Janua**(Figure 1): 40–49.

Norouzzadeh, M. S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M., Packer, C. and Clune, J. (2017). Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning, (1): 1–10.

Verma, G. K. and Gupta, P. (2018). *Proceedings of 2nd International Conference on Computer Vision & Image Processing*, Vol. 704, Springer Singapore.