Configuration Manual

MSc Research Project
MSc Data Analytics

Sayan Kumar Ghosh
Student ID: x22211781@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Teerath Kumar Menghwar

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. …Sayan Kumar Ghosh…………………………………………

Student ID: ……………x22211781@student.ncirl.ie…………………………………..……

Programme: …Msc Data analytics…………………………………… **Year:** …2023-2024..

Module: ………………Msc Research Project…………………………………………….………

Lecturer: ……………… Teerath Kumar Menghwar…………………………………….………

Submission Due Date: …………………12/08/2024…………………………………………………….………

Project Title: Improving Public Safety: Advanced Machine Learning for Early Detection of Aggressive Street Dog Behaviors in Asian Urban Environments

Word Count: …1191…………………… **Page Count:** ……………9……………………..……..………

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: ……………………Sayan Kumar Ghosh………

Date: ……………11/08/2024……………………………………….

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

Sayan Kumar Ghosh
Student ID: x22211781@student.ncirl.ie

1. Introduction

This paper focuses on the preview of dog emotion detection at a young age using deep learning techniques such as 2D-CNN, Vision Transformer, ResNeXt-50, and EfficientNet B3. The objective of the proposed research is to evaluate the efficiency of the above models in identifying and categorizing various emotional conditions in dogs from images.

2. System Configuration

The implementation of this project was carried out on Google Colab, utilizing its cloud-based servers. The system configuration included:
- CPU: Intel(R) Xeon(R) CPU @ 2.00GHz
- GPU: Tesla T4 with 2496 cores and 15GB DDR5 VRAM
- RAM: 51GB available
- Disk Space: 201GB available

3. Data Collection

The data set applied in this research was obtained from Kaggle[1] and arrived in images of dogs expressing various feelings which included anger, sadness, and even happiness. The first dataset included about 15000 images, and 4500 samples were chosen for this project to determine the emotional state of dogs per the project's targets and constraints in terms of computational capacity. Such selection process created basis for focused work on a large but not overwhelmingly large number of samples which is crucial for detailed analysis and building of effective models.

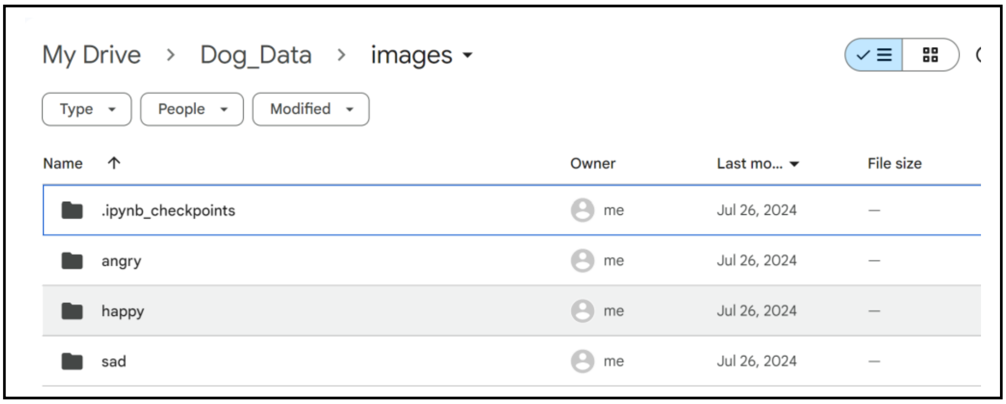

Figure 1

[1] https://www.kaggle.com/datasets/devzohaib/dog-emotions-prediction

4. Environment Setup

The dataset was downloaded, unzipped, and uploaded to Google Drive to ensure accessibility and ease of execution across different machines. Google Colab's predefined code was used to mount Google Drive, ensuring data security and efficient processing.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

    import zipfile
    import os

    # Define the path to your zip file
    zip_file_path = '/content/drive/MyDrive/Dog_Data.zip'

    # Define the extraction directory (same as the zip file's directory)
    extraction_dir = os.path.dirname(zip_file_path)

    # Unzip the file
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extraction_dir)

    # Verify the contents
    extracted_files = os.listdir(extraction_dir)
    print(extracted_files)
```

Figure 2

5. Data Exportation

All necessary Python libraries, including TensorFlow, PyTorch, and other auxiliary libraries, were imported to facilitate model training and evaluation.

Dog Detection

```
import os
import cv2
import torch
import json
import numpy as np
from tqdm.notebook import tqdm

# Load YOLOv5 model with GPU
model = torch.hub.load('ultralytics/yolov5', 'yolov5s').cuda()  # Use GPU

# Function to process images in batches
def process_images(folder_path, batch_size=32):
    results = []
    for i in tqdm(range(0, len([f for f in os.listdir(folder_path) if f.endswith(".jpg") or f.endswith(".png")]), batch_size)):
        batch_files = [f for f in os.listdir(folder_path) if f.endswith(".jpg") or f.endswith(".png")][i:i+batch_size]
        batch_images = [cv2.imread(os.path.join(folder_path, f)) for f in batch_files]

        # Perform detection
        results_model = model(batch_images)
        for j, detection in enumerate(results_model.xyxy):
            for det in detection.cpu().numpy():  # Ensure the tensor is on the CPU before processing
```

Figure 3

6. Exploratory Data Analysis

For this context, two techniques identified as Dog Detection and Dog Tracking are applied In this project to process image of dogs in different emotions including anger, sadness, and happiness. First, in the Dog Detection step, an image is forwarded into a machine learning model called YOLOv5 for detecting the Dog. This model processes the images in the batches, identifies the dogs and the coordinates and the degree of assurance of the identification into a JSON file. These include coordinates of the bounding box around the dog and a score representing the model's confidence that the object recognized as a dog.

After the detection, there is the Dog Tracking step executed in the proposed system. This is done with the help of a tracking algorithm known as DeepSORT which then helps in tracking the motion of each of the detected dogs in the frames of the images. From the bounding boxes coming from the detection results, it modifies the position of each dog to track the continuous movement of the dog in the video. The tracking results are also stored in a JSON file as the path a particular dog takes and the feelings of the dog during the sequence. These steps are necessary to observe the behaviors or possible links between dogs' movement and certain emotions.

```
# Clone the YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5.git
%cd yolov5

# Install the necessary dependencies
!pip install -r requirements.txt
```

Figure 4

Dog Detection

```
import os
import cv2
import torch
import json
import numpy as np
from tqdm.notebook import tqdm

# Load YOLOv5 model with GPU
model = torch.hub.load('ultralytics/yolov5', 'yolov5s').cuda()  # Use GPU

# Function to process images in batches
def process_images(folder_path, batch_size=32):
    results = []
    for i in tqdm(range(0, len([f for f in os.listdir(folder_path) if f.endswith(".jpg") or f.endswith(".png")]), batch_size)):
        batch_files = [f for f in os.listdir(folder_path) if f.endswith(".jpg") or f.endswith(".png")][i:i+batch_size]
        batch_images = [cv2.imread(os.path.join(folder_path, f)) for f in batch_files]

        # Perform detection
        results_model = model(batch_images)
        for j, detection in enumerate(results_model.xyxy):
            for det in detection.cpu().numpy():  # Ensure the tensor is on the CPU before processing
```

Figure 5

Dog Tracking

```
import os
import cv2
import json
from tqdm.notebook import tqdm
from deep_sort_realtime.deepsort_tracker import DeepSort

# Initialize DeepSORT
tracker = DeepSort(max_age=30, nn_budget=70, nms_max_overlap=1.0)

# Function to process detection results for tracking
def process_detections_for_tracking(detection_results):
    tracked_results = []
    for emotion, detections in detection_results.items():
        for detection in tqdm(detections, desc=f"Processing {emotion} images"):
            image_path = detection["image_path"]
            img = cv2.imread(image_path)
            img_height, img_width = img.shape[:2]

            # Extract bounding box and confidence score
            bbox = detection["bounding_box"]
            confidence = detection["confidence_score"]
```

Figure 6

7. Data Pre-processing

The dataset was split into training, validation, and test sets using a predefined ratio. This code is used to pre-process images of dogs for a machine learning model, which aims to identify such emotions as anger, sadness, or happiness. It loads tracking results, then resizes images and their formats for the model, and normalizes them to get better performance. A function takes each image and preprocesses it with the help of parallel processing. The emotions are assigned numerical values and the output data is divided into training and validation set so that the new images can be effectively detected by machine. Last of all , the processed data is kept for use in training in order to simplify subsequent steps when training the model.

Pre-processing

```
import torch
import json
import os
from tqdm.notebook import tqdm
from torchvision import transforms
from PIL import Image
import cv2
from sklearn.model_selection import train_test_split
from joblib import Parallel, delayed

# Load tracking results
tracking_results_path = '/content/drive/MyDrive/Dog_Data/images/dog_tracking_results.json'
with open(tracking_results_path, 'r') as f:
    tracked_results = json.load(f)

# Define transformation for Applying Model
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to the size expected by the CNN
    transforms.ToTensor(),  # Convert images to tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize images
])
```

Figure 7

```python
# Get preprocessed images and their corresponding emotions
images, emotions = preprocess_images(tracked_results)

# Map emotions to integers
emotion_to_idx = {'angry': 0, 'sad': 1, 'happy': 2}
int_emotions = [emotion_to_idx[emotion] for emotion in emotions]

# Split data into training and validation sets
train_images, val_images, train_emotions, val_emotions = train_test_split(images, int_emotions, test_size=0.2, random_state=42)

# Save preprocessed data
preprocessed_data_path = '/content/drive/MyDrive/Dog_Data/images/Dog_preprocessed_data.pth'
torch.save({
    'train_images': train_images,
    'val_images': val_images,
    'train_emotions': train_emotions,
    'val_emotions': val_emotions
}, preprocessed_data_path)

print(f"Number of training images: {len(train_images)}")
print(f"Number of validation images: {len(val_images)}")
print(f"Preprocessed data saved to {preprocessed_data_path}")
```

Figure 8

8. Data Transformation

After pre-processing, the training dataset underwent data transformation, including random flips, normalization, and resizing to fit the input requirements of different models.

```python
# Define transformation for Applying Model
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to the size expected by the CNN
    transforms.ToTensor(),  # Convert images to tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize images
])

# Function to preprocess a single image
def preprocess_image(result):
    image_path = result["image_path"]
    emotion = result["emotion"]

    img = cv2.imread(image_path)
    if img is None:
        print(f"Warning: Could not read image {image_path}")
        return None, None

    img_pil = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    img_tensor = transform(img_pil)

    return img_tensor, emotion
```

Figure 9

9. Data Modelling

Three models were used in this research: 2D CNN, Vision Transformer (ViT), ResNeXt-50 and EfficientNet B3. Each model was trained with specific configurations and hyperparameters tailored to optimize performance on the emotion detection task. All the models were saved as .pth files in a designated folder.

```python
# Define a simple 2D CNN for emotion recognition
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Linear(256 * 28 * 28, 1024),  # Update the input size based on your image dimensions
            nn.ReLU(inplace=True),
            nn.Linear(1024, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
```

Figure 10

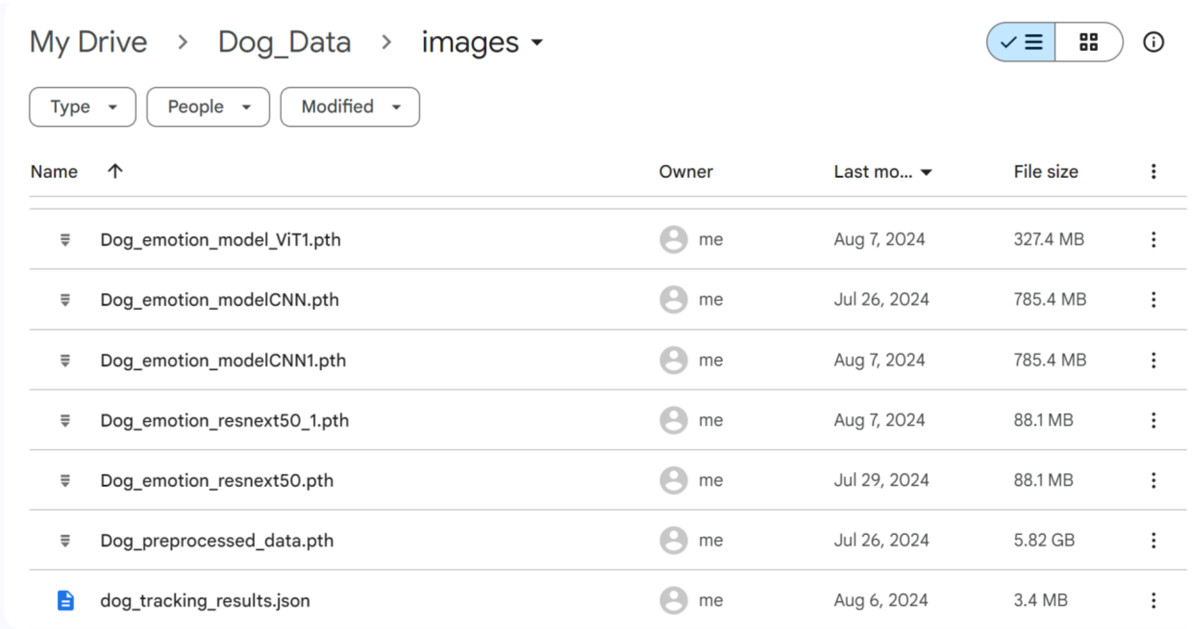

Figure 11

10. Model Training

Each model underwent extensive training with both pre-processed data. The best-performing models were saved for evaluation based on accuracy, loss, precision, recall, and F1 scores.

```
Epoch 3/5 [Training]: 100%   260/260 [00:52<00:00, 5.65it/s]
Epoch 3/5 [Validation]: 100%   65/65 [00:05<00:00, 11.05it/s]
Epoch 3/5, Train Loss: 0.6199, Train Accuracy: 0.7396, Val Loss: 0.2998, Val Accuracy: 0.8810
Model saved to /content/drive/MyDrive/Dog_Data/images/Dog_emotion_modelCNN1.pth after epoch 3
Epoch 4/5 [Training]: 100%   260/260 [00:51<00:00, 5.66it/s]
Epoch 4/5 [Validation]: 100%   65/65 [00:05<00:00, 11.33it/s]
Epoch 4/5, Train Loss: 0.1192, Train Accuracy: 0.9603, Val Loss: 0.1530, Val Accuracy: 0.9682
Model saved to /content/drive/MyDrive/Dog_Data/images/Dog_emotion_modelCNN1.pth after epoch 4
Epoch 5/5 [Training]: 100%   260/260 [00:51<00:00, 5.65it/s]
Epoch 5/5 [Validation]: 100%   65/65 [00:05<00:00, 11.53it/s]
Epoch 5/5, Train Loss: 0.0510, Train Accuracy: 0.9883, Val Loss: 0.1527, Val Accuracy: 0.9778
Model saved to /content/drive/MyDrive/Dog_Data/images/Dog_emotion_modelCNN1.pth after epoch 5
Training completed.
Accuracy: 0.9778420038535646
Precision: 0.9772967806249245
Recall: 0.978395349296005
F1 Score: 0.9778335637758877
Confusion Matrix:
[[869  10  14]
 [ 12 533   1]
 [  5   4 628]]
```

Figure 12

11. Model Evaluation

Evaluation Techniques included:

- Accuracy: Measures the proportion of correct predictions.
- Confusion Matrix: Displays the number of correct and incorrect predictions for each class.
- Precision and Recall: Precision measures the accuracy of positive predictions, while recall measures the ability to identify all positive instances.
- F1 Score: The harmonic mean of precision and recall, providing a single metric for model performance.

Results for 2D CNN, Vision Transformer (ViT), and ResNeXt-50 were detailed.

Conclusion

This study pointed out on how deep learning models are useful in recognizing emotions of a dog that are portrayed in a picture. The research also points to the fact that despite the models’ ability to produce the results satisfactory, the addition of a larger sample size, incorporating images of more diverse types and fine-tuning of parameters would lead to the increased model accuracy and performance. Thus, the need to develop better computational methods and strengthen data analysis to improve the efficiency of factors leading to emotion detection in real-life situations.