National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Wutyi Kyi Toe
Student ID: x23194286

School of Computing
National College of Ireland

Supervisor: Prof. Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Wutyi Kyi Toe……

Student ID: …… x23194286 ……..……

Programme: ……Artificial Intelligence…… **Year:** ……2024……..

Module: ……MSc Research Project……..……

Lecturer: …… Prof. Rejwanul Haque …….

Submission Due Date: ……12.12.2024.……

Project Title: Endangered Red Panda Behaviours Classification: A Comparative Evaluation of Deep Learning Models

Word Count: ……959…… **Page Count:** ……13……..……..

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: ……Wutyi Kyi Toe……

Date: ……12.12.2024.……

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

Wutyi Kyi Toe
Student ID: x23194286

1 Introduction

This paper presents the explanation of the coding procedure for the red panda behaviours classification project. The prerequisites and environment setup are also described so that any researcher who is interested in wildlife conservation can replicate and enhance this research project. Subsequently, the detailed steps for obtaining the dataset, required libraries, source codes and scripts, as well as the implementation of the deep learning models are elucidated.

2 Setup and Configuration

2.1. Hardware Configurations

The pre-processing for cropping images is processed on Windows 11 computer that has specifications of 3.00 GHz and Intel Core i7 processor with 16 GB RAM. Moreover, preinstalled Microsoft Visual Code version 1.95.3 is utilised.

2.2. Software Configurations

Python programming version 3.10.12 is utilised not only for preprocessing the images to be cropped and removing the unnecessary parts, such as trees, branches, and walls but also for building and training CNN models.

2.3. Prerequisites of Cloud Environment for Training Models

Google Colab Pro Version and Google Drive are used for training models and data storage to be streamlined without interference during training. Moreover, T4 GPUs with 15 GB of RAM are chosen in Google Colab for model training.

2.4 Libraries Requirements

Not only for checking similar indices while extracting the images from video footage and cropping the images but also for model training, the following libraries are required.

- TensorFlow 2.17.1
- Keras 3.5.0
- opencv 4.10.0.84
- numpy
- matplotlib
- sklearn
- seaborn

3 Dataset Preparation

3.1. Custom Data Collection

As the red panda dataset is not available publicly, both the video footage and images are recorded in Dublin Zoo as well as collected from the photographers (for red pandas) and from Japan and Red Panda Network.

3.2. Data Pre-processing

Firstly, calculating the Structural Similarity Index (SSIM) of the images while they are being extracted from video footage so that the images are not duplicated in the dataset. The source code is as shown as in Figure 1.

```
import cv2
import os
from skimage.metrics import structural_similarity as ssim
from google.colab import drive

drive.mount('/content/drive')

#Set paths for videos and output folders (execute and extract category by category, example below is for Eating category)
panda_videos_path = '/content/drive/MyDrive/Red Panda Project/Japan Zoo (To organize videos)/Eating'
unique_frames_path = '/content/drive/MyDrive/Red Panda Project/RP_Codes/Unique Panda/Eating'
duplicate_frames_path = '/content/drive/MyDrive/Red Panda Project/RP_Codes/Duplicate Panda'

os.makedirs(unique_frames_path, exist_ok=True)
os.makedirs(duplicate_frames_path, exist_ok=True)


ssim_threshold = 0.97  #SSIM threshold
desired_fps = 0.5 #1 frame per every two seconds


def is_similar(frame1, frame2, threshold=0.97):
    #Convert to grayscale
    grayA = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    grayB = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    #Compute SSIM between two frames to check
    score, _ = ssim(grayA, grayB, full=True)
    return score > threshold

for video_file in os.listdir(panda_videos_path):
    video_path = os.path.join(panda_videos_path, video_file)

    if video_path.endswith(('.mp4', '.avi', '.mov', '.mkv')):
        print(f"Processing video: {video_path}")
        video = cv2.VideoCapture(video_path)

        #Original FPS of the videos
        original_fps = video.get(cv2.CAP_PROP_FPS)
        frame_interval = int(original_fps / desired_fps)
```

Figure 1: Calculating SSIM from Extracted Images of Video Footage

After the images are extracted, the unnecessary parts from each image are cropped in Figure 2, as well as the images without red pandas are also removed.

```python
import cv2
import os

#Source and destination directories for resting (two other behaviours will be executed separately)
source_dir = r"C:\Users\wutyi\Documents\Red Panda\Japan\Resting"
dest_dir = r"C:\Users\wutyi\Documents\Red Panda\Japan\Cropped\Resting"

os.makedirs(dest_dir, exist_ok=True)

#Downscale factor
scale_factor = 0.5

#Function to manually crop an image with ROI selector of OpenCV
def manual_crop(image):

    downscaled_width = int(image.shape[1] * scale_factor)
    downscaled_height = int(image.shape[0] * scale_factor)
    downscaled_img = cv2.resize(image, (downscaled_width, downscaled_height), interpolation=cv2.INTER_AREA)

    r = cv2.selectROI("Select ROI is (Press Enter to skip)", downscaled_img, fromCenter=False, showCrosshair=True)

    if r[2] == 0 or r[3] == 0:
        return None

    x, y, w, h = [int(coord / scale_factor) for coord in r]

    #Using the adjusted coordinates
    cropped_image = image[y:y+h, x:x+w]
    return cropped_image


for filename in os.listdir(source_dir):
    if filename.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
        img_path = os.path.join(source_dir, filename)
        image = cv2.imread(img_path)

        if image is None:
            print(f"Failed to load {img_path}")
            continue


        cropped_image = manual_crop(image)

        if cropped_image is not None:
            save_path = os.path.join(dest_dir, filename)
            cv2.imwrite(save_path, cropped_image)
            print(f"Saved cropped image to {save_path}")
        else:
            print(f"Skipping {filename}, no crop was made.")
```

Figure 2: Cropping Unnecessary Parts from the Extracted Images

3.3. Pre-processed Dataset

The images are labelled into three classes (eating, resting and walking) and prepared as a dataset to utilize for model training.
The following link is the pre-processed dataset stored on Google Drive.
https://drive.google.com/drive/folders/1RxYg28QlODT0viptQt3tdkpaA3f6-0c_?usp=sharing

4 Data Modelling

There are three Convolutional Neural Network (CNN) models that are trained and compared their accuracy and efficiency. The first two pre-trained CNN models are EfficientNetB0 and ResNet50 that are being customised and trained with the red panda dataset using the transfer learning method. The third CNN model is self-trained from scratch and named as "CNN-RedPanda." All models are trained using Google Colab on cloud environment.

4.1. Pre-trained CNN Models (EfficientNetB0 and ResNet50) with transfer learning

Although EfficientNetB0 and ResNet50 are trained individually, they have the same pre-processing and data transformation steps as well as the same architecture style. The following

steps are the detailed process of training EfficientNetB0 and ResNet50 using the transfer learning method.
In Figure 3, the necessary libraries are imported for EfficientNetB0 and ResNet50.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import callbacks
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.regularizers import l2
import numpy as np
import matplotlib.pyplot as plt
```

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

```
Mounted at /content/drive
```

Figure 3: Importing Library for Pre-trained Models

After that, the dataset is split into two, training and validation with 80\% and 20\% in Figure 4.

```
data_dir = "/content/drive/MyDrive/Red Panda Project/RP_Codes/Extracted and Checked"
image_size = (256, 256)
batch_size = 32

#Split red panda dataset into training and validation as 80% and 20%
train_dataset = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=image_size,
    batch_size=batch_size,
)

validation_dataset = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 4801 files belonging to 3 classes.
Using 3841 files for training.
Found 4801 files belonging to 3 classes.
Using 960 files for validation.
```

Figure 4: Splitting Dataset Before Training Pre-trained Models

In Figure 5, data augmentation is applied before building and training the EfficientNetB0 and ResNet50 models with transfer learning.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2],
    channel_shift_range=20.0,
    fill_mode='nearest'
)


train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(256, 256),
    batch_size=32,
    class_mode='sparse'
)

#To reduce the taken time to fetch data from memory
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
```

Figure 5: Data Augmentation for training data

In Figure 6 and 7, the model architectures of EfficientNetB0 and ResNet50 are shown.

```
#Load EfficientNetB0
base_model = tf.keras.applications.EfficientNetB0(
    include_top=False,
    weights='imagenet',
    input_shape=(256, 256, 3)
)

#Unfreeze the last 10 layers to be in training and freeze the rest
base_model.trainable = True
for layer in base_model.layers[:-10]:
    layer.trainable = False

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(3, activation='softmax')  #3 classes(eating, resting, walking) of red pandas
])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Figure 6: Model Architecture of EfficientNetB0

```python
#Load pre-trained ResNet50 model
base_model = tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(256, 256, 3)
)

#Unfreeze the last 10 layers to be in training and freeze the rest
base_model.trainable = True
for layer in base_model.layers[:-10]:
    layer.trainable = False

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(3, activation='softmax')  #for eating, resting, walking behaviours of red pandas
])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Figure 7: Model Architecture of ResNet50

Subsequently, model training of EfficientNetB0 and ResNet50 are shown in Figure 8 and 9.

```python
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=3, restore_best_weights=True, verbose=1
)

history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    class_weight=class_weights_dict,
    callbacks=[early_stopping]
)

loss, accuracy = model.evaluate(validation_dataset)
print(f"Validation accuracy: {accuracy*100:.2f}%")
```

```
Epoch 1/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 300s 2s/step - accuracy: 0.6455 - loss: 5.3951 - val_accuracy: 0.9385 - val_loss: 4.4908
Epoch 2/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 108s 626ms/step - accuracy: 0.9043 - loss: 4.3177 - val_accuracy: 0.9708 - val_loss: 4.0267
Epoch 3/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 68s 563ms/step - accuracy: 0.9464 - loss: 3.9408 - val_accuracy: 0.9760 - val_loss: 3.6846
Epoch 4/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 90s 629ms/step - accuracy: 0.9657 - loss: 3.6528 - val_accuracy: 0.9729 - val_loss: 3.4130
Epoch 5/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 68s 561ms/step - accuracy: 0.9736 - loss: 3.3673 - val_accuracy: 0.9771 - val_loss: 3.1610
Epoch 6/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 89s 618ms/step - accuracy: 0.9838 - loss: 3.0949 - val_accuracy: 0.9792 - val_loss: 2.9176
Epoch 7/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 81s 608ms/step - accuracy: 0.9839 - loss: 2.8401 - val_accuracy: 0.9771 - val_loss: 2.6676
Epoch 8/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 84s 623ms/step - accuracy: 0.9880 - loss: 2.5937 - val_accuracy: 0.9833 - val_loss: 2.4360
Epoch 9/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 78s 649ms/step - accuracy: 0.9931 - loss: 2.3646 - val_accuracy: 0.9854 - val_loss: 2.2211
Epoch 10/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 78s 617ms/step - accuracy: 0.9916 - loss: 2.1526 - val_accuracy: 0.9833 - val_loss: 2.0194
Epoch 11/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 76s 573ms/step - accuracy: 0.9944 - loss: 1.9467 - val_accuracy: 0.9875 - val_loss: 1.8283
Epoch 12/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 88s 620ms/step - accuracy: 0.9926 - loss: 1.7653 - val_accuracy: 0.9833 - val_loss: 1.6531
```

Figure 8: Model Training of EfficientNetB0

```
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=3, restore_best_weights=True, verbose=1
)

history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    class_weight=class_weights_dict,
    callbacks=[early_stopping]
)

loss, accuracy = model.evaluate(validation_dataset)
print(f"Validation accuracy: {accuracy*100:.2f}%")
```

```
Epoch 1/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 480s 4s/step - accuracy: 0.7262 - loss: 5.3283 - val_accuracy: 0.9719 - val_loss: 4.3259
Epoch 2/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 120s 728ms/step - accuracy: 0.9710 - loss: 4.2064 - val_accuracy: 0.9740 - val_loss: 3.8822
Epoch 3/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 87s 716ms/step - accuracy: 0.9909 - loss: 3.7296 - val_accuracy: 0.9906 - val_loss: 3.4085
Epoch 4/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 84s 697ms/step - accuracy: 0.9957 - loss: 3.2876 - val_accuracy: 0.9875 - val_loss: 2.9887
Epoch 5/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 83s 688ms/step - accuracy: 0.9971 - loss: 2.8749 - val_accuracy: 0.9844 - val_loss: 2.6147
Epoch 6/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 145s 710ms/step - accuracy: 0.9952 - loss: 2.5007 - val_accuracy: 0.9875 - val_loss: 2.2558
Epoch 7/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 90s 738ms/step - accuracy: 0.9988 - loss: 2.1542 - val_accuracy: 0.9927 - val_loss: 1.9365
Epoch 8/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 133s 668ms/step - accuracy: 0.9972 - loss: 1.8550 - val_accuracy: 0.9917 - val_loss: 1.6626
Epoch 9/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 89s 731ms/step - accuracy: 0.9992 - loss: 1.5814 - val_accuracy: 0.9917 - val_loss: 1.4246
Epoch 10/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 81s 666ms/step - accuracy: 0.9986 - loss: 1.3498 - val_accuracy: 0.9927 - val_loss: 1.2161
Epoch 11/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 82s 668ms/step - accuracy: 0.9999 - loss: 1.1432 - val_accuracy: 0.9948 - val_loss: 1.0191
Epoch 12/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 87s 713ms/step - accuracy: 0.9995 - loss: 0.9655 - val_accuracy: 0.9927 - val_loss: 0.8671
Epoch 13/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 86s 713ms/step - accuracy: 0.9993 - loss: 0.8140 - val_accuracy: 0.9948 - val_loss: 0.7360
Epoch 14/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 87s 717ms/step - accuracy: 0.9967 - loss: 0.6923 - val_accuracy: 0.9948 - val_loss: 0.6357
Epoch 15/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 86s 714ms/step - accuracy: 0.9964 - loss: 0.5994 - val_accuracy: 0.9885 - val_loss: 0.5546
```

Figure 9: Model Training of ResNet50

4.2. Self-trained CNN Model (CNN-RedPanda)

In this subsection, the steps of data splitting, augmentation, model architecture and model training of CNN-RedPanda will be explained step by step with figures.

In Figure 10, the necessary libraries are imported for CNN-RedPanda.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import callbacks
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.regularizers import l2
import numpy as np
import matplotlib.pyplot as plt
```

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

```
Mounted at /content/drive
```

Figure 10: Importing Library for CNN-RedPanda

After that, the dataset is split into two, training and validation with 80\% and 20\% in Figure 11.

```
data_dir = "/content/drive/MyDrive/Red Panda Project/RP_Codes/Extracted and Checked"
image_size = (256, 256)
batch_size = 32

#Split red panda dataset into training and validation as 80% and 20%
train_dataset = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=image_size,
    batch_size=batch_size,
)

validation_dataset = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 4801 files belonging to 3 classes.
Using 3841 files for training.
Found 4801 files belonging to 3 classes.
Using 960 files for validation.
```

Figure 11: Splitting Dataset Before Training CNN-RedPanda

Figure 12 show the data augmentation methods are applied before CNN-RedPanda is built and trained.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    rotation_range= 20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2],
    channel_shift_range=30.0,
    fill_mode='nearest'
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(256, 256),
    batch_size=32,
    class_mode='sparse'
)

#To reduce the taken time to fetch data from memory
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
```

Figure 12: Data Augmentation for CNN-RedPanda Model

In Figure 13, the model architecture of CNN-RedPanda is shown.

```python
from tensorflow.keras import layers, models

#Custom CNN model
def create_custom_model(input_shape):
    model = models.Sequential()


    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))  #Fully connected layer with 256 units
    model.add(layers.Dropout(0.5))   #Dropout layer to reduce overfitting
    model.add(layers.Dense(3, activation='softmax'))  #Output layer with softmax activation for 3 behavioural classes of Red Panda (eating, resting, walking)
    return model


input_shape = (256, 256, 3)
model = create_custom_model(input_shape)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Figure 13: Model Architecture of CNN-RedPanda

Subsequently, model training of CNN-RedPanda is shown in Figure 14.

```python
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=3, restore_best_weights=True, verbose=1
)

history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    class_weight=class_weights_dict,
    callbacks=[early_stopping]
)

loss, accuracy = model.evaluate(validation_dataset)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 193s 2s/step - accuracy: 0.4609 - loss: 39.4684 - val_accuracy: 0.6396 - val_loss: 0.7323
Epoch 2/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 86s 621ms/step - accuracy: 0.6682 - loss: 0.7187 - val_accuracy: 0.7646 - val_loss: 0.5672
Epoch 3/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 64s 534ms/step - accuracy: 0.7727 - loss: 0.5416 - val_accuracy: 0.8104 - val_loss: 0.5134
Epoch 4/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 82s 533ms/step - accuracy: 0.8415 - loss: 0.3957 - val_accuracy: 0.8490 - val_loss: 0.4282
Epoch 5/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 93s 623ms/step - accuracy: 0.8847 - loss: 0.3188 - val_accuracy: 0.8979 - val_loss: 0.3236
Epoch 6/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 67s 555ms/step - accuracy: 0.8871 - loss: 0.2999 - val_accuracy: 0.8885 - val_loss: 0.3654
Epoch 7/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 75s 620ms/step - accuracy: 0.9209 - loss: 0.2235 - val_accuracy: 0.9042 - val_loss: 0.3442
Epoch 8/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 72s 537ms/step - accuracy: 0.9317 - loss: 0.1784 - val_accuracy: 0.9135 - val_loss: 0.3148
Epoch 9/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 75s 616ms/step - accuracy: 0.9445 - loss: 0.1466 - val_accuracy: 0.9135 - val_loss: 0.2919
Epoch 10/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 73s 600ms/step - accuracy: 0.9630 - loss: 0.1234 - val_accuracy: 0.9104 - val_loss: 0.3244
Epoch 11/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 73s 602ms/step - accuracy: 0.9614 - loss: 0.1174 - val_accuracy: 0.9219 - val_loss: 0.3531
Epoch 12/20
121/121 ━━━━━━━━━━━━━━━━━━━━ 83s 611ms/step - accuracy: 0.9577 - loss: 0.1354 - val_accuracy: 0.9177 - val_loss: 0.3226
Epoch 12: early stopping
Restoring model weights from the end of the best epoch: 9.
30/30 ━━━━━━━━━━━━━━━━━━━━ 12s 391ms/step - accuracy: 0.9082 - loss: 0.2855
Validation Accuracy: 91.35%
```

Figure 14: Model Training of CNN-RedPanda

5 Evaluation

The results of EfficientNetB0, ResNet50 and CNN-RedPanda are discussed in evaluation section. The results are explained with plots such as accuracy and loss curves of training and validation, confusion matrix, and classification report.

5.1. Evaluation for EfficientNetB0

In Figure 15, the curves show the accuracy and loss of EfficientNetB0 model during training and validation.

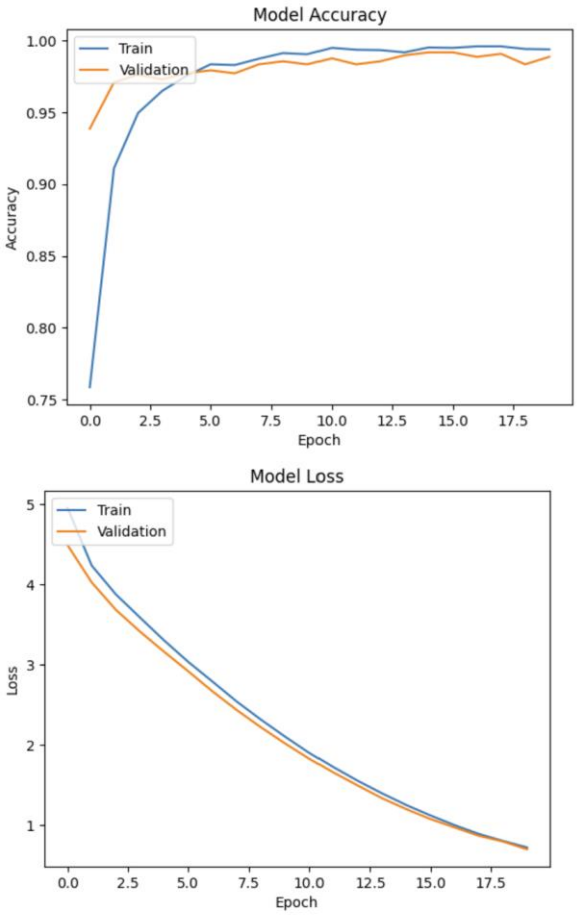
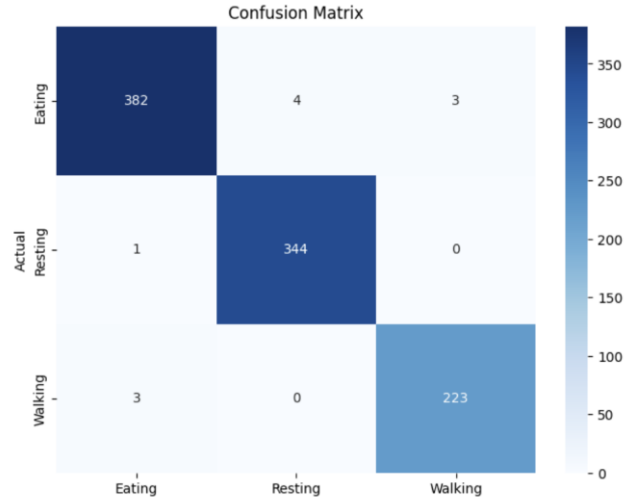


Figure 15: EfficientNetB0 Model Accuracy and Loss Curves

Figure 16 presents the confusion matrix and classification report of EfficientNetB0.



	precision	recall	f1-score	support
Eating	0.99	0.98	0.99	389
Resting	0.99	1.00	0.99	345
Walking	0.99	0.99	0.99	226
accuracy			0.99	960
macro avg	0.99	0.99	0.99	960
weighted avg	0.99	0.99	0.99	960

Figure 16: Confusion Matrix and Classification Report of EfficientNetB0 Model

5.2. Evaluation for ResNet50

In Figure 17, the curves show the accuracy and loss of ResNet50 model during training and validation.

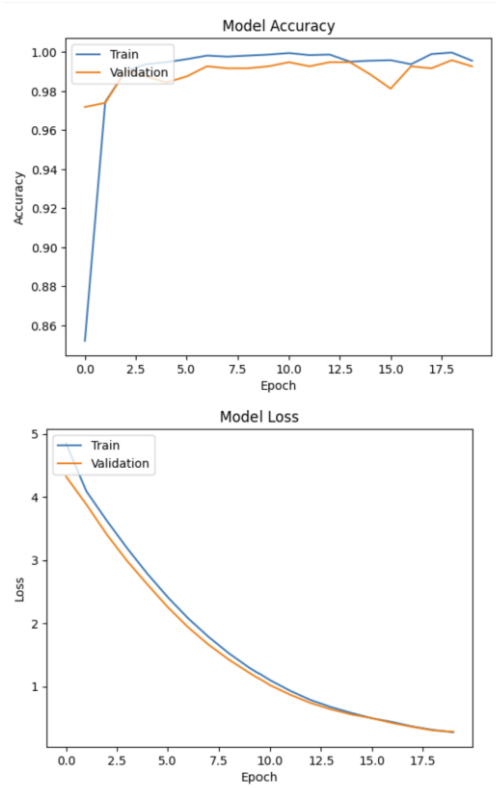
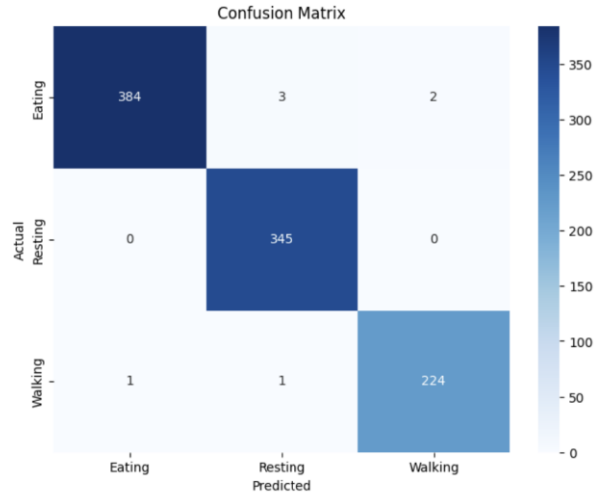


Figure 17: Accuracy and Loss Curves for ResNet50 Model

Figure 18 shows the confusion matrix and classification report of ResNet50.



	precision	recall	f1-score	support
Eating	1.00	0.99	0.99	389
Resting	0.99	1.00	0.99	345
Walking	0.99	0.99	0.99	226
accuracy			0.99	960
macro avg	0.99	0.99	0.99	960
weighted avg	0.99	0.99	0.99	960

Figure 18: Confusion Matrix and Classification Report of ResNet50 Model

4.3. Evaluation for CNN-RedPanda

In Figure 19, the curves show the accuracy and loss of CNN-RedPanda model during training and validation.

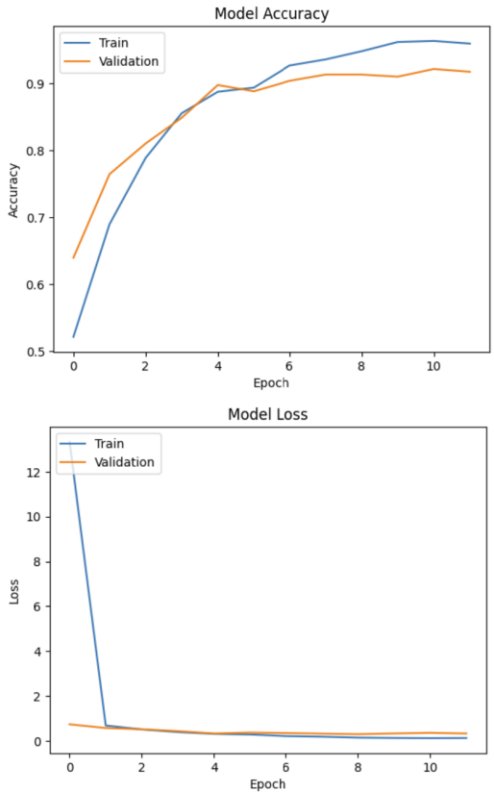
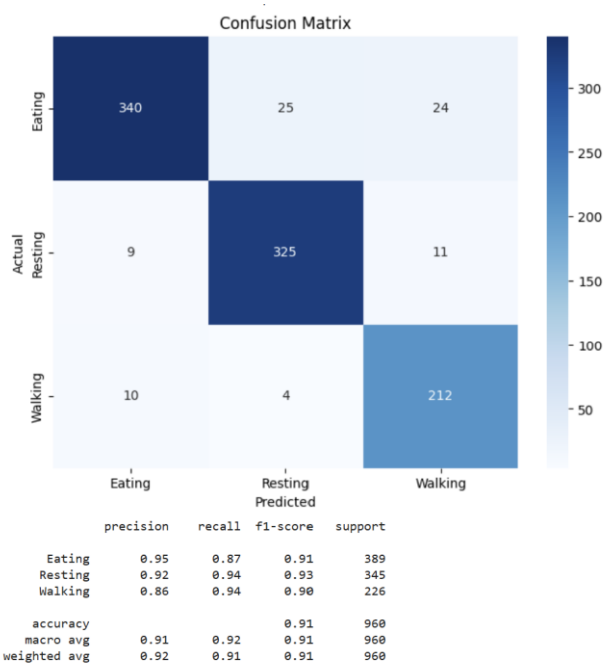


Figure 19: CNN-RedPanda Model Accuracy and Loss Curves

Figure 20 shows the confusion matrix and classification report of CNN-RedPanda.



	precision	recall	f1-score	support
Eating	0.95	0.87	0.91	389
Resting	0.92	0.94	0.93	345
Walking	0.86	0.94	0.90	226
accuracy			0.91	960
macro avg	0.91	0.92	0.91	960
weighted avg	0.92	0.91	0.91	960

Figure 20: Confusion Matrix and Classification Report of CNN-RedPanda Model