National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Munevver Irem Hatipoglu
Student ID: x23154861

School of Computing
National College of Ireland

Supervisor: Kislay Raj

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Munevver Irem Hatipoglu
Student ID:	x23154861
Programme:	Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Kislay Raj
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1320
Page Count:	19

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:	M. Irem Hatipoglu
Date:	8th August 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

Munevver Irem Hatipoglu
x23154861

1 Introduction

The goal of the Artificial Intelligence Based Psychological Prediction of HTP Drawings[1] project is to analyse the psychological conditions such as depression and anxiety of children by using various artificial intelligence algorithms to data collected from the House-Tree-Person (HTP) test, one of the most well-known art therapy methods. This manual provides detailed instructions for users who wish to execute the code, including setting up the project environment and making changes to the existing architecture as needed.

2 System Requirements

For effective completion of the project, several kinds of hardware and software features must be available. Users who do not have access to these features may have difficulty running the code. In this section, the technologies and tools used in the project are described in detail.

2.1 Hardware Requirements

Detailed hardware information of the device on which the project was developed is given in Table 1.

Hardware Features	The Author's Devide
Model	MacBook Pro
Chip	Apple M2
Cores	8 (4 performance and 4 efficiency)
RAM	8 GB
Hard Disk	256 GB
Operating System	MacOS 14.5

Table 1: Hardware Specifications

[1]Artificial Intelligence Based Psychological Prediction of HTP Drawings GitHub Repository: `https://github.com/iremhttp/HTP-Project`

2.2 Software Requirements

The project was developed with the open-source Python programming language and ran in the Jupyter Notebook environment via Anaconda Navigator. For this reason, the first step that needs to be taken in order for the program to be run on another device is to download the Anaconda Navigator[2] program. After the download process, the Jupyter Notebook should be launched from the Anaconda Navigator screen, which has an environment similar to Figure 1.

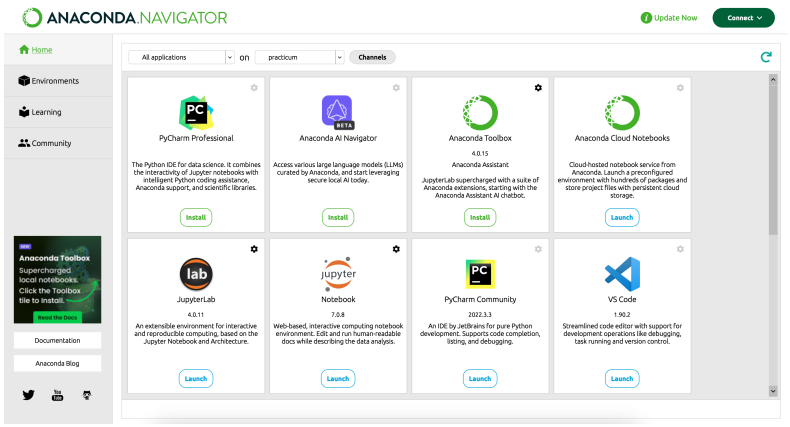

Figure 1: Anaconda Navigator

Different technological tools have been used in order to successfully implement the various steps carried out within the scope of the project. These tools and libraries are classified according to their intended use and are shown in Figure 2.

Figure 2: Libraries and Technologies Used

All the technologies and libraries used in the project have been imported as shown in Figure 3.

[2]Anaconda Navigator: `https://www.anaconda.com/products/navigator`

```
import os
import pandas as pd
import fitz
from PIL import Image
import shutil
import time
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import random
import cv2
from sklearn.utils import shuffle
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from collections import Counter
from sklearn.preprocessing import StandardScaler
import shutil
import subprocess
from quickdraw import QuickDrawDataGroup
from quickdraw import QuickDrawData
import tensorflow as tf
import keras
import keras.utils
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG16, ResNet50, EfficientNetB0
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.preprocessing import image
import keras.utils as image
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, classification_report, confusion_matrix
from keras import layers
import zipfile
import os
import json
from pycocotools.coco import COCO
```

Figure 3: Imported Libraries

3 Dataset Description & Loading

The main data set used in this project was created by the Psychology Department of Istanbul Bilgi University, which considered ethical values. The dataset contains PDF files representing each patient and containing HTP test drawings of the patient represented by the file. Additionally, it contains an Excel file with psychological scores assigned to these drawings by expert psychologists, as well as patient information. The dataset was used in this project thanks to the submission of special permission documents to Istanbul Bilgi University. Due to confidentiality rules, the dataset cannot be shared.

The mentioned data set is manually labelled as described in detail in the project report and downloaded in COCO format. This COCO file is uploaded to the system as shown in Figure 4 .

```
zip_file_path = 'son.zip'
extract_to = 'coco'

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to)

print("Zip file extracted successfully.")
```

```
Zip file extracted successfully.
```

Figure 4: Loading The Labelled Dataset

4 Dataset Restructuring

After the downloaded COCO zip file has been uploaded to the system, the dataset has been reshaped as shown in Figure 5 and 6 for easier use.

```
coco_folder = 'coco'
htp_images_folder = 'HTPImages'

images_source = os.path.join(coco_folder, 'images')
images_destination = os.path.join(htp_images_folder, 'images')

if os.path.exists(images_source):
    if not os.path.exists(images_destination):
        os.makedirs(images_destination)
    for filename in os.listdir(images_source):
        src_file = os.path.join(images_source, filename)
        dst_file = os.path.join(images_destination, filename)
        shutil.move(src_file, dst_file)
    print(f"Moved images folder to {htp_images_folder}.")
else:
    print(f"{images_source} does not exist.")

results_file_source = os.path.join(coco_folder, 'result.json')
results_file_destination = os.path.join(htp_images_folder, 'result.json')

if os.path.exists(results_file_source):
    shutil.move(results_file_source, results_file_destination)
    print(f"Moved results.json to {htp_images_folder}.")
else:
    print(f"{results_file_source} does not exist.")

if os.path.exists(coco_folder):
    shutil.rmtree(coco_folder)
    print(f"Deleted the {coco_folder} folder.")
else:
    print(f"{coco_folder} folder does not exist.")
```

Figure 5: Moving Files In COCO File

```
images_subfolder = os.path.join(htp_images_folder, 'images')

for filename in os.listdir(htp_images_folder):
    file_path = os.path.join(htp_images_folder, filename)

    if os.path.isfile(file_path) and filename not in ['images', 'result.json']:
        shutil.move(file_path, os.path.join(images_subfolder, filename))

print("All images moved to the 'images' subfolder successfully.")
```

Figure 6: Moving Images To The ”Image” Folder

The JSON file that comes with the COCO file and contains the tag and object location information of the images has been updated to match the images that have just been moved to the ”images” file, as can be seen in Figure.

```python
coco_annotation_file = 'HTPImages/result.json'
images_folder = 'HTPImages/images'

with open(coco_annotation_file, 'r') as f:
    coco_data = json.load(f)

for image_info in coco_data['images']:
    base_filename = image_info['file_name'].split('-')[-1]

    image_info['file_name'] = base_filename

with open(coco_annotation_file, 'w') as f:
    json.dump(coco_data, f, indent=4)

print("File names in results.json have been updated successfully.")

missing_files = []
for image_info in coco_data['images']:
    image_path = os.path.join(images_folder, image_info['file_name'])
    if not os.path.exists(image_path):
        missing_files.append(image_info['file_name'])

if missing_files:
    print("The following image files are missing in the images folder:")
    for missing_file in missing_files:
        print(missing_file)
else:
    print("All images are present in the images folder.")
```

Figure 7: Updating JSON File

5 Dataset Preprocessing & Augmentation

The images in the dataset were cropped using object location information from the JSON file as shown in Figure 8. It provided that unrelated sketches drawn by children on the same paper were eliminated from the dataset to avoid them influencing the decision-making process of the project's models.

```python
image_base_path = 'HTPImages/images'

for ann in annotations['annotations']:
    image_id = ann['image_id']
    bbox = ann['bbox']
    category_id = ann['category_id']
    category_name = category_dict[category_id]

    img_filename = image_dict[image_id]
    img_path = os.path.join(image_base_path, img_filename)

    img = cv2.imread(img_path)
    if img is None:
        print(f"Warning: Image {img_filename} could not be loaded.")
        continue

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    #Extract bounding box
    x, y, w, h = bbox

    # Crop the image using the box
    cropped_img = img[int(y):int(y + h), int(x):int(x + w)]
    cv2.imwrite(img_path, cropped_img)

print("Images have been cropped and saved successfully.")
```

Figure 8: Cropping Images

Then, the images went through the resizing and normalisation preprocessing steps as in Figures 9 and 10.

```python
# RESIZE
image_folder = 'HTPImages/images'
target_size = (224, 224)

for filename in os.listdir(image_folder):
    file_path = os.path.join(image_folder, filename)
    if os.path.isfile(file_path) and filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        img = cv2.imread(file_path)
        resized_img = cv2.resize(img, target_size)
        cv2.imwrite(file_path, resized_img)
print("Resizing completed and images updated in place.")
```

Figure 9: Resizing Images

```python
# NORMALIZE
def normalize_image(img):
    return img / 255.0

for filename in os.listdir(image_folder):
    file_path = os.path.join(image_folder, filename)
    if os.path.isfile(file_path) and filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        img = cv2.imread(file_path)
        normalized_img = normalize_image(img)
        normalized_img = (normalized_img * 255).astype(np.uint8)
        cv2.imwrite(file_path, normalized_img)

print("Normalization completed and images updated in place.")
```

Figure 10: Normalising Images

After these operations, the data-frame, where the data in the Excel file in the original dataset was previously moved and updated for easier use, as in Figure 11, has gone through various preparatory processes as in Figure 12 and 13.

```python
xlsx_path = 'HTPDataset/patient_info.xlsx'
df_excel = pd.read_excel(xlsx_path)
df_excel.head()
```

Figure 11: Loading Excel From Original Dataset

```python
with open('HTPImages/result.json', 'r') as f:
    annotations = json.load(f)

image_dict = {img['file_name']: img['id'] for img in annotations['images']}
category_dict = {cat['id']: cat['name'] for cat in annotations['categories']}
annotation_dict = {ann['image_id']: category_dict[ann['category_id']] for ann in annotations['annotations']}
```

```python
filename_to_label = {}
for image in annotations['images']:
    file_name = image['file_name']
    image_id = image['id']
    if image_id in annotation_dict:
        filename_to_label[file_name] = annotation_dict[image_id]
```

```python
df['label'] = df['image_file'].map(filename_to_label)

print("Label column added to DataFrame and saved successfully.")
```

Figure 12: Creating Column For Labels

In the next step, the empty files that are not labelled in the dataset are removed as Figure .

```
df = df.dropna(subset=['label'])

print(df.info())
```

Figure 13: Drop Images With No Labels

```
image_folder = 'HTPImages/images'

labelled_images = df['image_file'].tolist()

for filename in os.listdir(image_folder):
    if filename not in labelled_images:
        file_path = os.path.join(image_folder, filename)
        os.remove(file_path)
        print(f"Removed unlabelled image: {filename}")

remaining_images = os.listdir(image_folder)
print(f"Number of remaining labelled images: {len(remaining_images)}")
```

Figure 14: Remove Images With No Labels

Figure 15 and 16 shows the creation of a separate dataset file for use in the feature extraction section, which is one of the important steps of the project, and the data augmentation process performed using the QuickDraw dataset.

```
base_dir = "HTPImages"
feature_dir = os.path.join(base_dir, "feature")

categories = {
    "house": QuickDrawDataGroup("house"),
    "tree": QuickDrawDataGroup("tree"),
    "face": QuickDrawDataGroup("face")
}

for category in categories.keys():
    folder_name = "person" if category == "face" else category
    subfolder_path = os.path.join(feature_dir, folder_name)
    os.makedirs(subfolder_path, exist_ok=True)

for category, data_group in categories.items():
    folder_name = "person" if category == "face" else category
    folder_path = os.path.join(feature_dir, folder_name)

    for i in range(500):
        drawing = data_group.get_drawing()
        image = drawing.image.convert("RGB")
        image_path = os.path.join(folder_path, f"{category}_{i+1}.jpg")
        image.save(image_path, "JPEG")

    print(f"500 {category} images from QuickDraw have been saved to '{folder_path}' in JPEG format.")

print("All images have been saved.")
```

Figure 15: Feature Extraction Dataset Creation and Data Augmentation

After the completion of the formation of the dataset for feature extraction, the images were subjected to preprocessing steps as can be seen in Figure 17.

```
base_dir = "HTPImages"
images_folder = os.path.join(base_dir, "images")
feature_dir = os.path.join(base_dir, "feature")
annotations_file = os.path.join(base_dir, "result.json")

for category_name in category_dict.values():
    subfolder_path = os.path.join(feature_dir, category_name)
    os.makedirs(subfolder_path, exist_ok=True)

for annotation in annotations['annotations']:
    image_id = annotation['image_id']
    category_id = annotation['category_id']
    category_name = category_dict[category_id]

    image_info = next(img for img in annotations['images'] if img['id'] == image_id)
    image_filename = image_info['file_name']
    image_path = os.path.join(images_folder, image_filename)

    dest_path = os.path.join(feature_dir, category_name, image_filename)

    shutil.copy(image_path, dest_path)

print("All images have been copied to the appropriate subfolders in the 'feature' directory.")
```

Figure 16: Feature Extraction Dataset Preparation

```
target_size = (224, 224)

def preprocess_image(img_path, target_size):
    img = Image.open(img_path)
    img = img.resize(target_size)
    img_array = np.array(img) / 255.0
    return img_array
```

Figure 17: Feature Extraction Dataset Preprocessing

As shown in Figure 18, this prepared dataset has gone through the Class Weights and Label Encoder steps to be ready for model training.

```
def get_images(directory):
    images = []
    labels = []

    directory = directory if directory.endswith(os.path.sep) else directory + os.path.sep

    for label in os.listdir(directory):
        label_path = os.path.join(directory, label)

        if os.path.isdir(label_path):
            for image_file in os.listdir(label_path):
                image_path = os.path.join(label_path, image_file)

                if image_file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    image = cv2.imread(image_path)
                    image = cv2.resize(image, (224, 224))
                    image = image / 255.0

                    images.append(image)
                    labels.append(label)

    return shuffle(images, labels, random_state=817328462)
```

```
images, labels = get_images("./HTPImages/feature/")

images = np.array(images)
labels = np.array(labels)

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)

X_train, X_test, y_train, y_test = train_test_split(images, encoded_labels, test_size=0.2, random_state=42)

label_counts = Counter(encoded_labels)
print("Label distribution:", label_counts)

class_weights = compute_class_weight('balanced', classes=np.unique(encoded_labels), y=encoded_labels)
class_weight_dict = {i: weight for i, weight in enumerate(class_weights)}
print("Class Weights:", class_weight_dict)
```

Figure 18: Class Weights and Label Encoder

6 Model Implementation & Evaluation

6.1 Feature Extraction Models

The first model used for feature extraction has a structure similar to the ResNet50 Figure 19.

```
resnet_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = resnet_base.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

resnet_model = Model(inputs=resnet_base.input, outputs=predictions)

for layer in resnet_base.layers[:-10]:
    layer.trainable = False

for layer in resnet_base.layers[-10:]:
    layer.trainable = True

resnet_model.compile(optimizer=Adam(learning_rate=1e-5), loss=SparseCategoricalCrossentropy(), metrics=["accuracy"])

resnet_model.summary()
```

Figure 19: ResNet50 Architecture

The model trained by using the code in Figure

```
start_time_resnet = time.time()
training_history_resnet = resnet_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, class_weight=class_weight_dict)
train_time_resnet = time.time() - start_time_resnet
```

Figure 20: ResNet50 Training

The ResNet50 model is evaluated as shown in Figure 21.

```
start_time_resnet = time.time()
y_pred = np.argmax(resnet_model.predict(X_test), axis=1)
predict_time_resnet = time.time() - start_time_resnet

f1 = f1_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')

print("F1 Score:", f1)
print("Recall:", recall)
print("Precision:", precision)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Training Time:", train_time_resnet)
print("Prediction Time:", predict_time_resnet)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='rocket', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for ResNet Model')
plt.show()
```

Figure 21: ResNet50 Evaluation

The architecture of the VGG16 model, the second model used for this task, is created as shown in Figure 22.

```
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = vgg_base.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

vgg_model = Model(inputs=vgg_base.input, outputs=predictions)

for layer in vgg_base.layers[:-10]:
    layer.trainable = False

for layer in vgg_base.layers[-10:]:
    layer.trainable = True

vgg_model.compile(optimizer=Adam(learning_rate=1e-5), loss=SparseCategoricalCrossentropy(), metrics=["accuracy"])

vgg_model.summary()
```

Figure 22: VGG16 Architecture

The training of the model is as shown in Figure 23.

```
start_time_vgg = time.time()
training_history_vgg = vgg_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, class_weight=class_weight_dict)
train_time_vgg = time.time() - start_time_vgg
```

Figure 23: VGG16 Training

Figure 24 represents the evaluation of the model.

```
start_time_vgg = time.time()
y_pred = np.argmax(vgg_model.predict(X_test), axis=1)
predict_time_vgg = time.time() - start_time_vgg

f1 = f1_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')

print("F1 Score:", f1)
print("Recall:", recall)
print("Precision:", precision)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Training Time:", train_time_vgg)
print("Prediction Time:", predict_time_vgg)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='rocket', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for VGG16 Model')
plt.show()
```

Figure 24: VGG16 Evaluation

The latest model applied for feature extraction, EfficientNet, is created as in Figure 25.

```
efficientnet_base = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = efficientnet_base.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

efficientnet_model = Model(inputs=efficientnet_base.input, outputs=predictions)

for layer in efficientnet_base.layers[:-10]:
    layer.trainable = False

for layer in efficientnet_base.layers[-10:]:
    layer.trainable = True

efficientnet_model.compile(optimizer=Adam(learning_rate=1e-5), loss=SparseCategoricalCrossentropy(), metrics=["accuracy"])

efficientnet_model.summary()
```

Figure 25: EfficientNet Architecture

Figure 26 shows the code for training the model.

```
start_time_effnet = time.time()
training_history_efficientnet = efficientnet_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, class_weight=class_weigh
train_time_effnet = time.time() - start_time_effnet
```

Figure 26: EfficientNet Training

Finally, the model performance was evaluated as shown in Figure 27.

```
start_time_effnet = time.time()
y_pred = np.argmax(efficientnet_model.predict(X_test), axis=1)
predict_time_effnet = time.time() - start_time_effnet

f1 = f1_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')

print("F1 Score:", f1)
print("Recall:", recall)
print("Precision:", precision)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Training Time:", train_time_effnet)
print("Prediction Time:", predict_time_effnet)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='rocket', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for EfficientNet Model')
plt.show()
```

Figure 27: EfficientNet Evaluation

The performance measurements of all models and their comparison with each other are represented in the code block in Figure 28.

```python
def evaluate_model(model, X_test, y_test):
    y_pred = np.argmax(model.predict(X_test), axis=-1)
    print(classification_report(y_test, y_pred))

print("VGG16 Model Performance")
evaluate_model(vgg_model, X_test, y_test)

print("ResNet50 Model Performance")
evaluate_model(resnet_model, X_test, y_test)

print("EfficientNet Model Performance")
evaluate_model(efficientnet_model, X_test, y_test)
```

Figure 28: Models' Performances

The relationship between the epoch and the accuracy-loss change of the models was examined with the code block in Figure 29.

```python
training_histories = {
    'VGG16': training_history_vgg,
    'ResNet50': training_history_resnet,
    'EfficientNetB0': training_history_efficientnet
}
```

```python
def plot_training_history(histories, metric='accuracy'):
    plt.figure(figsize=(12, 8))

    for model, history in histories.items():
        plt.plot(history.history[metric], label=f'{model} Training {metric.capitalize()}')
        plt.plot(history.history[f'val_{metric}'], label=f'{model} Validation {metric.capitalize()}')

    plt.title(f'Model {metric.capitalize()}')
    plt.xlabel('Epoch')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.grid(True)
    plt.show()
```

```python
plot_training_history(training_histories, metric='accuracy')
```

Figure 29: Models' Accuracy and Loss By Epochs

Due to its best performance, the VGG16 model was saved as shown in Figure 30 to extract the features of the images in the following stages of the project.

```python
model_save_path = "vgg_model.h5"
vgg_model.save(model_save_path)
```

```python
loaded_model = load_model(model_save_path)
```

Figure 30: Model Save and Load

After examining the models, it was seen that the ResNet50 and VGG16 models showed high accuracy performance. Therefore, the decision-making processes of these models were analysed using the Grad-CAM approach. For this purpose, functions are defined first (Figure 31).

```python
def get_img_array(img_path, size):
    img = image.load_img(img_path, target_size=size)
    array = image.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = keras.models.Model(
        model.inputs, [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

Figure 31: Grad-CAM Functions Defined

Then, as in Figure 32, an image was randomly selected from the dataset to be examined with the Grad-CAM.

```python
dataset_dir = "./HTPImages/feature"
image_files = []
for root, dirs, files in os.walk(dataset_dir):
    for file in files:
            image_files.append(os.path.join(root, file))

random_image_path = random.choice(image_files)

img_array = get_img_array(random_image_path, size=(224, 224))
```

Figure 32: Image Selection for Grad-CAM

First, the code block in Figure 33 was applied to analyse the features that the ResNet50 model pays attention to when making decisions.

```python
heatmap = make_gradcam_heatmap(img_array, resnet_model, "conv5_block3_out")

heatmap_expanded = np.expand_dims(heatmap, axis=(0, -1))
heatmap_resized = tf.image.resize(heatmap_expanded, (img_array.shape[1], img_array.shape[2]))

heatmap_resized = tf.squeeze(heatmap_resized)

original_img = image.img_to_array(image.load_img(random_image_path)) / 255.0

plt.figure(figsize=(5, 5))
plt.imshow(original_img, alpha=0.6)
plt.imshow(heatmap_resized, cmap='jet', alpha=0.4)
plt.title("Grad-CAM Heatmap Overlay")
plt.axis("off")
plt.show()
```

Figure 33: ResNet50 Grad-CAM

Finally, the VGG16 model was examined through Grad-CAM (Figure 34).

```
vgg_model = VGG16(weights='imagenet', include_top=True)
heatmap = make_gradcam_heatmap(img_array, vgg_model, "block5_conv3")

heatmap_expanded = np.expand_dims(heatmap, axis=(0, -1))
heatmap_resized = tf.image.resize(heatmap_expanded, (img_array.shape[1], img_array.shape[2]))

heatmap_resized = tf.squeeze(heatmap_resized)

original_img = image.img_to_array(image.load_img(random_image_path)) / 255.0

plt.figure(figsize=(5, 5))
plt.imshow(original_img, alpha=0.6)
plt.imshow(heatmap_resized, cmap='jet', alpha=0.4)
plt.title("Grad-CAM Heatmap Overlay")
plt.axis("off")
plt.show()
```

Figure 34: VGG16 Grad-CAM

6.2 Psychological Detection Models

Following the Feature Extraction experiments, the final stage of the project started, which is Psychological Detection. In order to make psychological detection easier, the psychological scores found in the data-frame were first examined. After the boundaries and average values of these scores were carefully examined with the code in Figure 35, the thresholds found.

```
columns_of_interest = ['externalising_score', 'anxious_score', 'depressive_score']

for column in columns_of_interest:
    if column in df.columns:
        min_value = df[column].min()
        max_value = df[column].max()
        mean_value = df[column].mean()
        median_value = df[column].median()

        print(f"Statistics for {column}:")
        print(f"  Min Value: {min_value}")
        print(f"  Max Value: {max_value}")
        print(f"  Mean Value: {mean_value:.2f}")
        print(f"  Median Value: {median_value:.2f}")
        print()
    else:
        print(f"Column '{column}' not found in DataFrame.")
```

Figure 35: Analysing the Scores

Then, the numerical scores in the data-frame were converted into categorical scores (Figure 36).

In the next step, the dataset that will be used in the experiment to detect psychological disorders such as depression and anxiety, has been reconstructed in light of the changes in the data-frame. First, sub-folders were created in the dataset as in Figure 37.

```python
def categorize_scores(row):
    # Categorize Externalising Score
    if row['externalising_score'] <= 9:
        row['externalising_category'] = 'non_externalised'
    else:
        row['externalising_category'] = 'externalised'

    # Categorize Anxious Score
    if row['anxious_score'] <= 6:
        row['anxious_category'] = 'non_anxious'
    else:
        row['anxious_category'] = 'anxious'

    # Categorize Depressive Score
    if row['depressive_score'] <= 4:
        row['depressive_category'] = 'non_depressed'
    else:
        row['depressive_category'] = 'depressed'

    return row
```

```python
df = df.apply(categorize_scores, axis=1)
df = df.drop(columns=['externalising_score', 'anxious_score', 'depressive_score'])
df
```

Figure 36: Updating Data-frame By Using The Thresholds Determined

```python
base_dir = 'HTPImages/images'
categories = {
    'depression': ['non_depressed','depressed'],
    'anxiety': ['non_anxious', 'anxious'],
    'externalise': ['non_externalised','externalised']
}

for category, subfolders in categories.items():
    category_path = os.path.join(base_dir, category)
    os.makedirs(category_path, exist_ok=True)
    for subfolder in subfolders:
        subfolder_path = os.path.join(category_path, subfolder)
        os.makedirs(subfolder_path, exist_ok=True)
        print(f"Folder '{subfolder}' created at '{subfolder_path}'")
```

Figure 37: Sub-folder Creation

After that, the images were moved to the created sub-folders according to the psychological characteristics they contained (Figure 38).

In the psychological detection task, depression detection experiments were first conducted. The VGG16 model, which was saved because of its performance in Feature Extraction experiments, was used to extract the characteristics of the "Depression" data (Figure 39).

```python
for index, row in df.iterrows():
    image_file = row['image_file']
    image_path = os.path.join(base_dir, image_file)

    if not os.path.isfile(image_path):
        print(f"Image {image_file} not found in the directory. Skipping...")
        continue

    depressive_category = row['depressive_category']
    anxious_category = row['anxious_category']
    externalising_category = row['externalising_category']

    destinations = {
        'depression': os.path.join(base_dir, 'depression', depressive_category),
        'anxiety': os.path.join(base_dir, 'anxiety', anxious_category),
        'externalise': os.path.join(base_dir, 'externalise', externalising_category)
    }

    for key, dest_path in destinations.items():
        dest_image_path = os.path.join(dest_path, image_file)
        shutil.copy2(image_path, dest_image_path)

    os.remove(image_path)

print("Image organization completed.")
```

Figure 38: Dataset Organised

```python
def get_features(model, image_path, target_size=(224, 224)):
    img = image.load_img(image_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    features = model.predict(img_array)
    return features.flatten()

feature_extraction_model = Model(inputs=loaded_model.inputs,
                                 outputs=loaded_model.get_layer("block5_pool").output)

image_dir = 'HTPImages/images/depression'

feature_vectors = []
image_files = []
labels = []

for root, dirs, files in os.walk(image_dir):
    for file in files:
        if file.lower().endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(root, file)
            features = get_features(feature_extraction_model, image_path)
            feature_vectors.append(features)
            image_files.append(file)

            if 'non_depressed' in root:
                labels.append('non_depressed')
            else:
                labels.append('depressed')
```

Figure 39: Feature Extraction for Depression

Then, the results obtained as shown were stored in a special data-frame created for easier use as can be seen in Figure 40.

```python
features_dict = {image_file: feature for image_file, feature in zip(image_files, feature_vectors)}

depression_df = df[['image_file', 'depressive_category']].copy()

depression_df['features'] = depression_df['image_file'].map(features_dict)
depression_df
```

Figure 40: Depression Data-frame Creation

Due to the unbalanced distribution of the data, the Class Weights approach was applied as shown in Figure 41.

```python
def extract_features(df):
    return np.array([features for features in df['features']])

X = extract_features(depression_df)
y = depression_df['depressive_category']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_encoded),
    y=y_encoded
)
class_weight_dict = dict(enumerate(class_weights))
```

Figure 41: Depression Dataset Imbalance Handled by Class Weights

Using the common machine learning function shown in Figure 42, multiple models were trained with HTP image features. Their performance was then assessed by testing.

```python
def ml_models(model, X, y, class_weight=None):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    predict_time = time.time() - start_time

    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)

    report = classification_report(y_test, y_pred, target_names=label_encoder.classes_)
    print("Classification Report:\n", report)

    print(f"Training Time: {train_time:.4f} seconds")
    print(f"Prediction Time: {predict_time:.4f} seconds")

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

Figure 42: Common Depression Detection Function

The machine learning approaches proposed to be used in the project report are implemented as in Figure 43 using the common function.

```python
#RANDOM FOREST
rf_model = RandomForestClassifier(n_estimators=300, random_state=42, max_depth=300, class_weight=class_weight_dict)
ml_models(rf_model, X, y_encoded)

# DECISION TREE
dt_model = DecisionTreeClassifier()
ml_models(dt_model, X, y_encoded)

# GAUSSIAN NB
gnb_model = GaussianNB()
ml_models(gnb_model, X, y_encoded)

# SVM
svm_model = SVC(kernel='linear', random_state=42, class_weight=class_weight_dict)
ml_models(svm_model, X, y_encoded)

#LOGISTIC REGRESSION
lr_model = LogisticRegression(max_iter=1000, random_state=42, class_weight=class_weight_dict)
ml_models(lr_model, X, y_encoded)
```

Figure 43: Model Training and Evaluation By Common Function

Another psychological aspect explored in the model's performance detection is anxiety. First, as in the depression detection experiment, the features of the images in the anxiety-specific dataset are extracted using the saved model (Figure 44).

```
feature_extraction_model = Model(inputs=loaded_model.inputs,
                                 outputs=loaded_model.get_layer("block5_pool").output)

image_dir = 'HTPImages/images/anxiety'

feature_vectors = []
image_files = []
labels = []

for root, dirs, files in os.walk(image_dir):
    for file in files:
        if file.lower().endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(root, file)
            features = get_features(feature_extraction_model, image_path)
            feature_vectors.append(features)
            image_files.append(file)

            if 'non_anxious' in root:
                labels.append('non_anxious')
            else:
                labels.append('anxious')
```

Figure 44: Feature Extraction for Anxiety

Following this step, the extracted features are stored in a specially created data-frame, as in Figure 45.

```
features_dict = {image_file: feature for image_file, feature in zip(image_files, feature_vectors)}

anxiety_df = df[['image_file', 'anxious_category']].copy()

anxiety_df['features'] = anxiety_df['image_file'].map(features_dict)
anxiety_df
```

Figure 45: Anxiety Data-frame Creation

Figure 46 shows the Class Weights approach applied in order to prevent errors that imbalance in the dataset may cause in the decision process of the models.

```
def extract_features(df):
    return np.array([features for features in df['features']])

X = extract_features(anxiety_df)
y = anxiety_df['anxious_category']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_encoded),
    y=y_encoded
)
class_weight_dict = dict(enumerate(class_weights))
```

Figure 46: Anxiety Dataset Imbalance Handled by Class Weights

Using a common machine learning function, the models are trained, tested, and performance information is provided for accurate analysis (Figure 47).

```python
def ml_models(model, X, y, class_weight=None):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    predict_time = time.time() - start_time

    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)

    report = classification_report(y_test, y_pred, target_names=label_encoder.classes_)
    print("Classification Report:\n", report)

    print(f"Training Time: {train_time:.4f} seconds")
    print(f"Prediction Time: {predict_time:.4f} seconds")

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='flare', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

Figure 47: Common Anxiety Detection Function

Finally, the effectiveness of various machine learning models in detecting anxiety by HTP images with the defined function was examined as can be seen in Figure 48.

```python
#RANDOM FOREST
rf_model = RandomForestClassifier(n_estimators=300, random_state=42, max_depth=300, class_weight=class_weight_dict)
ml_models(rf_model, X, y_encoded)

# DECISION TREE
dt_model = DecisionTreeClassifier()
ml_models(dt_model, X, y_encoded)

# GAUSSIAN NB
gnb_model = GaussianNB()
ml_models(gnb_model, X, y_encoded)

# SVM
svm_model = SVC(kernel='linear', random_state=42, class_weight=class_weight_dict)
ml_models(svm_model, X, y_encoded)

#LOGISTIC REGRESSION
lr_model = LogisticRegression(max_iter=1000, random_state=42, class_weight=class_weight_dict)
ml_models(lr_model, X, y_encoded)
```

Figure 48: Model Training and Evaluation By Common Function