National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ian Dias
Student ID: 22205748

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes, Prof. Musfira Jilani

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Ian Dias
Student ID:	22205748
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Paul Stynes, Prof. Musfira Jilani
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1172
Page Count:	10

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:	Ian Dias
Date:	6th 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

Ian Dias
22205748

1 Introduction

This document will help to specify all preparations to be made to replicate the experiments conducted in the research, Section 2 will help understand the hardware and cloud resource specifications used for the experiment, Section 3 defines all steps to prepare data for the experiment and mention all libraries used. Section 3.3.1 shows the functions defined for the metrics calculations, the next sections 4 and 5 show the model training steps and evaluations.

2 Integrating Environment

This section shows via the 3 tables below the hardware, software, and cloud requirements recommended for this project.

Host Machine	Lenovo Ideapad 3
Processor	2.42 GHz Intel Core i5
Operating System	Windows 11
RAM	16.0 GB
SSD	512 GB

Table 1: Hardware Recommendations

Programming Language	Python 3.11
IDE	Jupyter Notebook
Browser	Brave,Chrome,Edge

Table 2: Software recommendations

Cloud Storage	Google Drive
Cloud Platform	Google Collab Pro
Cloud resource	TPU V2 high ram

Table 3: Cloud requirements

3 Data preparation

This section will be divided into three subsections which will describe the collection of data in section 3.1, preparing a sample from it in Section 3.2 and then perform processing steps on the data for further usage in Section 3.3.

3.1 Data Collection

This step involves gathering the data from external sources in order to conduct the experiments, for this research the author has requested access of the data from **Mr. Igor Kolesnikov**, who is the author of the paper **'Unveiling Galaxy Morphology through an Unsupervised-Supervised Hybrid Approach'** Kolesnikov et al. (2024), this data can also be accessed from the sdss MANGA DAP website `https://www.sdss4.org/dr17/manga/manga-data/catalogs/` and instructions to download the data can be found at the DAP documentation `https://sdss-mangadap.readthedocs.io/en/latest/metadatamodel.html#dapall-database` for the FITS data, and `https://zenodo.org/records/3565489#.Y3vFKS-l0eY` for the jpeg images and table data. But this process is rather complex, so it is recommended to get in touch with the author of previous works who can provide data.

3.2 Creating a Sample

In order to conduct tests it is recommended to create small samples from the data rather than running the code on the entire dataset.

To do so, the author follows two simple steps, the first is to manually copy a certain set of galaxy fits file data and store it in a folder and then run the python script shown in Fig 1 and 2 in order to get matching samples of the jpeg images. This step is only required to run the **"Combined Methodology"** experiments.

```python
import os
import shutil

def find_matching_files(source_folder, target_folder):
    source_files = os.listdir(source_folder)
    target_files = os.listdir(target_folder)
    results = {}
    i = 0
    for source_file in source_files:
        source_name, source_ext = os.path.splitext(source_file)
        found = False

        for target_file in target_files:
            target_name, target_ext = os.path.splitext(target_file)
            if source_name == target_name:
                results[source_file] = f"Found as {target_file}"
                found = True
                print(f"found{i}")
                i += 1
                break
        if not found:
            results[source_file] = "Not Found"
            os.remove(os.path.join(source_folder, source_file))

    return results

def print_results(results):
    print("File Search Results:")
    print("--------------------")
    for file_name, status in results.items():
        print(f"{file_name}: {status}")
```

Figure 1: Finding the matching file

```python
def copy_matching_files(source_folder, target_folder, destination_folder):
    os.makedirs(destination_folder, exist_ok=True)
    source_files = os.listdir(source_folder)
    target_files = os.listdir(target_folder)
    copied_count = 0

    for source_file in source_files:
        source_name, _ = os.path.splitext(source_file)
        for target_file in target_files:
            target_name, _ = os.path.splitext(target_file)
            if source_name == target_name:
                source_path = os.path.join(source_folder, source_file)
                target_path = os.path.join(target_folder, target_file)
                destination_path = os.path.join(destination_folder, target_file)
                shutil.copy2(target_path, destination_path)
                copied_count += 1
                break

    print(f"\nCopied {copied_count} files to {destination_folder}")
```

Figure 2: Copying Matching Files

To create a sample for the **"Supervised learning"** there is no need for the copying step above,but the requirement for this step is to assign labels to the images which can be done as shown in fig 3

```python
import glob
# Paths to videos
eliptical = glob.glob('/content/drive/MyDrive/Thesis/jpg_supervised/Eliptical/*.jpg')
spiral = glob.glob('/content/drive/MyDrive/Thesis/jpg_supervised/Spiral/*.jpg')
total = eliptical + spiral

# Label assignment
def get_labels(total):
    labels = {}
    for image in total:
        if "Eliptical" in image:
            labels[image] = 1
        elif "Spiral" in image:
            labels[image] = 0
    return labels

labels = get_labels(total)
```

Figure 3: Steps for assigning labels

3.3 Processing Data

This section shall give an insight on the necessary steps for preparing the data for the experiments. This section is divided into two subsections that talk about processing the FITS data and the JPEG images in sections 3.3.1 and 3.3.2, respectively.

3.3.1 FITS Data processing

This step involves calculating the metrics data from the images extracted from the FITS files Lotz et al. (2004). The functions for entropy and Gini are shown in figures 4 and 5, respectively.

The Cymorph package [1] can be used to calculate the gradient pattern (second moment).

In order to use this functions it is important to extract image data from the fits file and create a segmentation mask of the image to input into these funtions. Extracting the image data from the FITS file can be done using the astropy.io package [2] and further

[1] Cymorph: `https://cymorph.readthedocs.io/en/latest/metrics.html`
[2] Astropy :`https://github.com/astropy/astropy`

```python
def get_entropy(image_segmented, nbins):
    line = image_segmented[image_segmented!=0].flatten()

    freq = np.array([0.0 for i in range(nbins)], dtype=np.float32)
    counts, bins = np.histogram(line, nbins)

    somatorio = 0.0
    for i in range(nbins):
        somatorio = somatorio + counts[i]

    try:
        for i in range(nbins):
            freq[i] = float(counts[i]) / float(somatorio)
    except ZeroDivisionError:
        return np.nan

    somatorio = 0.0
    for i in range(nbins):
        if freq[i]>0.0:
            somatorio = somatorio - freq[i] * np.log10(freq[i])

    entropy_coeficient = somatorio/np.log10(nbins)

    return entropy_coeficient
```

Figure 4: Entropy Calculation function

```python
def get_gini(image, segmentation):
    vector = image[segmentation >= 1].flatten()
    vector = np.sort(vector)

    N = len(vector)

    denominator = np.mean(vector)*N*(N-1)

    numerator = 0
    for i in range(0,len(vector)):
        numerator = (numerator)  + (((2*i) - N - 1)*vector[i])

    Gini = numerator/denominator

    return(float(Gini))
```

Figure 5: Gini Calculation function

segmentation and bakground extraction can be done using the SEP package [3] as used in the research or SExtractor package [4] as an alternative.

The functions for creating a cutout and using these packages to create segmented mask are shown in figures 6 and 7.

[3]SEP: `https://github.com/kbarbary/sep`
[4]SExtractor: `https://sextractor.readthedocs.io/en/latest/Measurements.html`

```python
def create_cutout(image, center, cutout_size):
    y, x = center
    half_size = cutout_size // 2
    cutout = image[y-half_size:y+half_size, x-half_size:x+half_size]
    return cutout

def clean_secondary_objects(cutout, primary_mask):
    secondary_mask = ~primary_mask
    cutout[secondary_mask] = np.random.normal(np.mean(cutout[primary_mask]),
                                              np.std(cutout[primary_mask]),
                                              np.sum(secondary_mask))
    return cutout

def produce_segmentation_image(cutout, petrosian_radius):
    # Apply Gaussian smoothing to reduce noise
    smooth_cutout = gaussian_filter(cutout, sigma=4)


    # Calculate the threshold based on the intensity at the Petrosian radius
    threshold = np.mean(smooth_cutout[smooth_cutout > petrosian_radius])

    # Create the segmentation image
    segmentation_image = smooth_cutout > threshold


    return segmentation_image
```

Figure 6: Image Cutout and segmentation functions

```python
for filename in os.listdir(directory):
    if filename.endswith(('.fit', '.fits')):
        file_path = os.path.join(directory, filename)

        try:
          with fits.open(file_path) as hdul:
              # Assuming the image data is in the primary HDU
              # You might need to adjust this depending on the structure of your FITS files
              image_data = hdul[0].data

              # Step 2: Create a cutout of the galaxy
              center = (image_data.shape[0] // 2, image_data.shape[1] // 2) # Example center
              cutout_size = int(scale * petrosian_radius)
              cutout = create_cutout(image_data, center, cutout_size)

              # Step 3: Clean secondary objects
              # Assuming primary_mask is a binary mask of the primary galaxy
              primary_mask = cutout > np.median(cutout)  # Simplified mask creation for demonstration
              clean_cutout = clean_secondary_objects(cutout, primary_mask)

              # Step 4: Produce segmentation image
              segmentation_image = produce_segmentation_image(clean_cutout, petrosian_radius)

              data = np.ascontiguousarray(clean_cutout)
              data = data.byteswap().newbyteorder()
              bkg = sep.Background(data)

              data_sub = data - bkg
              objs, segmentedSep = sep.extract(data_sub, 1.5, err=bkg.globalrms,
                                               segmentation_map=True, maskthresh=1.5)

              mask = segmentedSep.astype('float32')
```

Figure 7: applying the segmentation flow

Once the segmentation mask is obtained for a specific FITS file, it can be used to calculate the metrics as shown in Figure 8

In a similar way, these steps can be followed for calculating the entropy and gradient moment as well.

```python
if image_data.ndim == 2:
    # Single image
    # clean_cutout = np.array(data_sub, dtype=np.float32)
    gini_coeff = get_gini(data_sub, mask)
    results[filename] = gini_coeff
elif image_data.ndim == 3:
    # Multiple images
    results[filename] = [get_gini(data_sub, mask) for image in image_data]
else:
    print(f"Unexpected data dimensions in {filename}")
```

Figure 8: Applying Metrics Function (Gini in this case)

3.3.2 Processing Jpeg images

Processing the jpeg images is simple and only requires resizing the images according to the requirements of the supervised model (128*128 for CNN, and 224 * 224 for EfficentNetB0) and further normalizing the image as shown in figure 9.

```python
# Parameters
img_height, img_width = 128, 128

# Function to load and preprocess images
def load_and_preprocess_image(filepath):
    img = load_img(filepath, target_size=(img_height, img_width))
    img_array = img_to_array(img)
    img_array = img_array / 255.0  # Normalize to [0, 1]
    return img_array

# Load and preprocess images
df['image'] = df['path'].apply(load_and_preprocess_image)
```

Figure 9: Jpeg processing example for CNN

Once the pre-processing is done, it is recommended to store the processed data as a pickle file as this step may take a lot of time and a considerable amount of RAM. The steps to store and retrieve the data as a pickle file are shown in figures 10 and 11.

```python
import pickle
data = df.copy()
# Convert tensor to numpy array
# data['image'] = data['image'].numpy()

# Save the DataFrame to a pickle file
with open('/content/drive/MyDrive/Thesis/Processed_CNN.pkl', 'wb') as f:
    pickle.dump(data, f)

print("DataFrame saved to 'Processed_CNN.pkl'")
```

Figure 10: Storing data as a pickle file

```python
import pickle

# Load the DataFrame from the pickle file
with open('/content/drive/MyDrive/Thesis/Processed_CNN.pkl', 'rb') as f:
    df_loaded = pickle.load(f)

# Convert the numpy array back to a tensor
# df_loaded['image'][0] = tf.convert_to_tensor(df_loaded['image'][0])

print("DataFrame loaded from 'dataframe_with_tensor.pkl'")
print(df_loaded)
```

Figure 11: Retreiving Pickle file

4 Model Training

This section will help to understand the parameters used for training the models used in the research. Sections 4.1 and 4.2 shall talk about the Clustering parameters and the cassification parameters respectively.

4.1 Clustering models

The first algorithm to be discussed is the SOM algorithm, for which the parameters **learning rate** and **sigma** parameters are set to **0.005** and **0.3**, respectively. And the grid size is calculated using the formula $grid\_size = \sqrt{(n\_subjects * 0.1)}$ Yaa et al. (2023). These are shown in figure 12

```python
# Set up and train the SOM
grid_size = int(round(math.sqrt(metrics_df.shape[0]*0.1), 0)) # REFERENCE FORMULA IN REPORT
som_dim = (grid_size, grid_size)  # 5x5 grid, adjust as needed
som = MiniSom(som_dim[0], som_dim[1], normalized_data.shape[1], sigma=0.3, learning_rate=0.005)
som.random_weights_init(normalized_data)
som.train_random(normalized_data, 1000)  # train for 1000 iterations

# Get cluster assignments
clusters = [som.winner(x) for x in numeric_data]
metrics_df['Cluster'] = [f"{w[0]},{w[1]}" for w in clusters]

# Print the first few rows of the result
print(metrics_df.head())
```

Figure 12: SOM parameters

The next algorithm is the HDBScan algorithm for which the min_cluster_size parameter has been set to 16, as shown in fig 13.

```python
clusterer = hdbscan.HDBSCAN(min_cluster_size=16).fit(normalized_data)
color_palette = sns.color_palette('deep', 8)
cluster_colors = [color_palette[x] if x >= 0
                  else (0.5, 0.5, 0.5)
                  for x in clusterer.labels_]
cluster_member_colors = [sns.desaturate(x, p) for x, p in
                         zip(cluster_colors, clusterer.probabilities_)]
```

Figure 13: Running HDBScan

Another additional step followed in this research, to get an estimate of the top hierarchy of clusters is to run the agglomerative clustering on the formulated clusters as shown in 14.

```
# 1. Calculate centroids for each cluster
clusters = data.groupby('Cluster')[['Gini', 'Entropy', 'G2']].mean()

# 2. Normalize the centroids
scaler = StandardScaler()
normalized_centroids = scaler.fit_transform(clusters)

# 3. Perform hierarchical clustering to get 2 superclusters
hierarchical_clustering = AgglomerativeClustering(n_clusters=2)
supercluster_labels = hierarchical_clustering.fit_predict(normalized_centroids)

# 4. Create a mapping from original clusters to superclusters
cluster_to_supercluster = dict(zip(clusters.index, supercluster_labels))

# 5. Assign superclusters to the original data
data['Supercluster'] = data['Cluster'].map(cluster_to_supercluster)
```

Figure 14: Agglomerative cluster to form top Hierarchy

4.2 Classification Models

Both models use the same parameters that are **"adam"** as the optimizer and **"entropy loss"** as the loss function, categorical entropy loss for EfficientNetB0 and binary cross entropy loss for CNN.

The major difference comes from the models architecture, the application of the models are shown in figures 15 and 16.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.losses import CategoricalCrossentropy

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
label_smoothing = 0.1  # Example value, can be adjusted
loss = CategoricalCrossentropy(label_smoothing=label_smoothing)
model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])
model.summary()
```

Figure 15: CNN model architecture

```python
from tensorflow.keras import layers
from tensorflow.keras.applications import EfficientNetB0

NUM_CLASSES = 2
IMG_SIZE = 224
size = (IMG_SIZE, IMG_SIZE)


inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))


# Using model without transfer learning

outputs = EfficientNetB0(include_top=True, weights=None,
                         classifier_activation="softmax",
                         pooling="max",
                         classes=NUM_CLASSES)(inputs)
```

Figure 16: EfficientNetB0 training

5 Evaluations

For evaluations, both algorithms are evaluated seperately as discussed in sections 5.1 and 5.2.

5.1 Cluster evaluations

To evaluate the performance of the clusters the generated labels are compared against the true labels to estimate how well the clusters perform in generating labels based on the calculated metrics, the function for this is shown in 17.

```python
# Merge DataFrames to keep only paths present in both
df1 = df1[df1['JPG'].isin(df3['JPG'])]

# Merge DataFrames on the 'path' column for comparison
merged_df1 = pd.merge(df1, df3, on='JPG', suffixes=('_True', '_HDB_ENet'))

# Compare the labels
matches = merged_df1['Supercluster_True'] == merged_df1['Supercluster_HDB_ENet']

# Calculate the percentage of matches
match_percentage = matches.mean() * 100

print(f"Percentage of matches: {match_percentage:.2f}%")

# Optional: If you want to see which rows match
merged_df1['matches'] = matches
print(merged_df1)
```

Figure 17: Evaluating cluster results

5.2 Classification Evaluations

For evaluating the classification performances the following methods are used, confusion Matrix, F1 score, precision, recall, etc. as shown in the figure below.

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict the labels for the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Compute the confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)

# Plot the confusion matrix
plt.figure(figsize=(8, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=np.unique(y_true_classes))
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title('Confusion Matrix')
plt.show()

# Calculate precision, recall, and F1 score
precision = precision_score(y_true_classes, y_pred_classes, average='weighted')
recall = recall_score(y_true_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_true_classes, y_pred_classes, average='weighted')

# Print the metrics
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

Figure 18: Classfication Evaluations

References

Kolesnikov, I., Sampaio, V. M., de Carvalho, R. R., Conselice, C., Rembold, S. B., Mendes, C. L. and Rosa, R. R. (2024). Unveiling Galaxy Morphology through an Unsupervised-Supervised Hybrid Approach, *Monthly Notices of the Royal Astronomical Society* **528**(1): 82–107. arXiv:2401.08906 [astro-ph].
URL: *http://arxiv.org/abs/2401.08906*

Lotz, J. M., Primack, J. and Madau, P. (2004). A New Nonparametric Approach to Galaxy Morphological Classification - IOPscience. Publisher: IOP Publishing.
URL: *https://iopscience.iop.org/article/10.1086/421849*

Yaa, Noirot Céline, H. J. M. J.-A. K. M. , Fanny, Guilmineau Camille, K. A. M. D. S. and Nathalie, Gaspin Christine, L. L. V. H.-S. (2023). *Section 11 Self-Organizing Map (SOM) | ASTERICS: User documentation.*
URL: *https://asterics.pages.mia.inra.fr/user*$_d$*ocumentation/som.htmlref* – *vialaneixWSOM*2017