Configuration Manual

MSc Research Project
Data Analytics

Ramsha Amir
Student ID: X23218215

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ramsha Amir
Student ID:	X23218215
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Aaloka Anant
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	8

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:	Ramsha Amir
Date:	28th January 2025

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

Ramsha Amir
X23218215

1 Introduction

This configuration manual walks through the different stages of code development for the Few-Shot Thoracic Disease Classification Using Prototypical Networks. These stages include:

- Data Collection
- Data Preprocessing
- Implementation of Few-Shot Thoracic Disease Classification Using Prototypical Networks
- Evaluation of Systems

2 System Configuration

2.1 Hardware Specification

A configuration manual contains comprehensive setup instructions for a system or device. The goal of the handbook is to fully describe how to carry out the research study. It also details the machine setup needed to create and execute the models. The procedures include installing the necessary apps and packages as well as the basic configuration advised for a project to be successful.

The system was implemented on Google co-labs. The detailed description of the environment is in the figure below

Machine Source	Google Colabs
Type	GPU
Name	Tesla T4
RAM	16GB

Table 1: Environment Specification

2.2 Software and Libraries

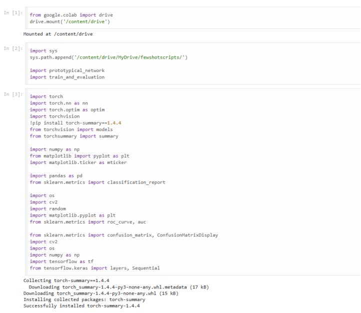
The list below contains the libraries and software tools used for this research.

- Python 3
- Torch==2.1.0
- Torchvision==0.16.0
- Torch-summary==1.4.4
- Numpy >=1.24.0
- Pandas >=2.00
- Matplotlib >=3.7.0
- Scikit-learn >=1.3.0
- Opencv-python >=4.8.0
- VGG19,ResNet50 and DenseNet121 Model Weights

Download DenseNet121 Model Link
Download ResNet50 Model Link
Download VGG19 Model Link

2.3 Importing the required Libararies

Figure 1 below shows the number of packages and libraries imported for the classification task.


```
In [1]: from google.colab import drive
        drive.mount('/content/drive')

Mounted at /content/drive

In [2]: import sys
        sys.path.append('/content/drive/MyDrive/fewshotscripts/')

        import prototypical_network
        import train_and_evaluation

In [3]: import torch
        import torch.nn as nn
        import torch.optim as optim
        import torchvision
        !pip install torch-summary==1.4.4
        from torchvision import models
        from torchsummary import summary

        import numpy as np
        from matplotlib import pyplot as plt
        import matplotlib.ticker as mticker

        import pandas as pd
        from sklearn.metrics import classification_report

        import os
        import cv2
        import random
        import matplotlib.pyplot as plt
        from sklearn.metrics import roc_curve, auc

        from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
        import cv2
        import os
        import numpy as np
        import tensorflow as tf
        from tensorflow.keras import layers, Sequential

Collecting torch-summary==1.4.4
  Downloading torch_summary-1.4.4-py3-none-any.whl.metadata (17 kB)
Downloading torch_summary-1.4.4-py3-none-any.whl (15 kB)
Installing collected packages: torch-summary
Successfully installed torch-summary-1.4.4
```

Figure 1: Importing the necessary python libraries

3 Data Collection

For the research 2 datasets were used NIH Chest X-ray Image Data Set which is available on kaggle has a size of 2.5GB. Another dataset of x-ray images of Covid19 was also utilised which is also available on kaggle has a size of 187MB.

```python
def read_xray_images(directory_path, img_height, img_width, selected_classes=None, augmentation=True,
                     normalize=False, apply_clahe_flag=True, display_comparison=False):
    datax = []
    datay = []

    sub_directories = os.listdir(directory_path)

    if selected_classes is None:
        selected_classes = sub_directories
    else:
        for cls in selected_classes:
            if cls not in sub_directories:
                raise ValueError(f"Class '{cls}' not found in {directory_path}. Available classes: {sub_directories}")

    for sub_dir in selected_classes:
        sub_dir_path = os.path.join(directory_path, sub_dir)
        correct_image_filepaths = check_image_path(sub_dir_path)

        for fpath in correct_image_filepaths:
            image = cv2.imread(fpath)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, (img_width, img_height))

            original_image = image.copy()

            if apply_clahe_flag:
                enhanced_image = apply_clahe(image)
                image = enhanced_image

            if augmentation:
                image_tensor = tf.convert_to_tensor(image, dtype=tf.float32)
                augmented_image = augmentation_layer(image_tensor)  # Apply augmentation layer
                augmented_image = augmented_image.numpy().astype(np.uint8)  # Convert back to numpy array

            if normalize:
                image = image / 255.0

            if display_comparison:
                display_image_comparison(original_image, enhanced_image, augmented_image)

            datax.append(image)
            datay.append(sub_dir)

    return np.array(datax), np.array(datay)
```

Figure 2: Read Images Functions

NIH Chest X-ray Dataset Link

Covid19 Dataset Link

4 Data Preprocessing

To train the dataset for this research different techniques were utilised. All the techniques of preprocessing applied on the data are as below:

- **Input Validation** Checks for valid image extensions (.png, .jpg, .jpeg) and verifies existence of specified classes in the directory
- **Image Resizing** All images are resized to a fixed size of 224×224×3 pixels (HEIGHT × WIDTH × CHANNELS).This standardization is crucial for deep learning models that expect consistent input dimensions
- **Color Space Conversion** Images are first read in BGR format (OpenCV default) Converted from BGR to RGB color space using cv2.
- **CLAHE** Contrast Limited Adaptive Histogram Equalization) Converts image to grayscale and applies CLAHE with clip limit of 2.0 and tile grid size of 8×8 then converts it back to RGB.It Helps improve contrast and enhance image details.
- **Random Horizontal Flipping** It Randomly flips images horizontally which helps model learn orientation-invariant features.
- **Random Zoom** Height factor range: -1% to +10% and Width factor range: -10% to +10% to adds robustness to scale variations

5 System Implementation

Starting by loading the pretrained models which will be our feature extractors

After loading we will freeze the model to stops any parameter updates during back-propagation and Flatten outputs from the Convolutional base networks and remove the Linear layers

```python
def apply_clahe(image, clip_limit=2.0, tile_grid_size=(8, 8)):
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_grid_size)
    enhanced_image = clahe.apply(gray_image)
    enhanced_image_rgb = cv2.cvtColor(enhanced_image, cv2.COLOR_GRAY2RGB)
    return enhanced_image_rgb
```

Figure 3: Clahe Function

```python
augmentation_layer = Sequential([
    layers.RandomFlip(mode='horizontal', seed=CFG.TF_SEED),
    layers.RandomZoom(height_factor=(-.01, 0.1), width_factor=(-0.1, 0.1), seed=CFG.TF_SEED)
], name='augmentation_layer')
```

Figure 4: Augmentation Layer

```python
def check_image_path(directory):
    return [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith(('.png', '.jpg', '.jpeg'))]
```

Figure 5: Image path

```python
for fpath in correct_image_filepaths:
    image = cv2.imread(fpath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (img_width, img_height))
```

Figure 6: Resize and Color Space Code

After that we will be doing episodic learning using prototypical network and calculate euclidean distance between feature vectors which can be seen in the figure below

Then we will be doing episodic learning with Prototypical Network on a test set and saves every 100th episode result to a CSV file.

6 Evaluation

Evaluation in the code was done using classification reports , confusion matrix and plotting ROC Curves. Predict Function to test samples can be seen in the figure 17 below

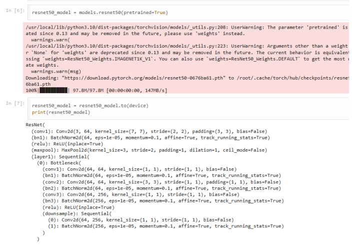


Figure 7: ResNet50 Model

Figure 8: DenseNet121 Model

Figure 9: VGG19 Model

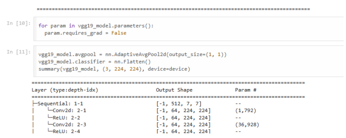


Figure 10: VGG19 Freeze And Flatten

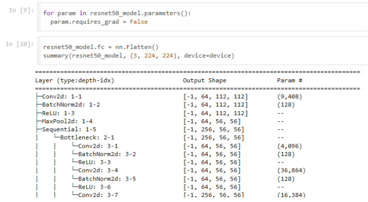


Figure 11: ResNet50 Freeze and Flatten

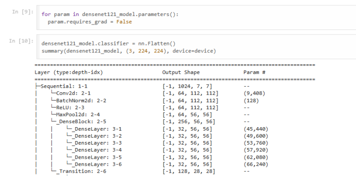


Figure 12: DenseNet121 Freeze and Flatten

```python
class ProtoNet(nn.Module):
    def __init__(self, encoder, device="CPU"):
        super(ProtoNet, self).__init__()
        self.device = device
        self.encoder = encoder.to(self.device)
    def set_forward_loss(self, sample):
        sample_images = sample['images'].to(self.device) # retrieve the support + query images
        n_way = sample['n_way']                 # get no. of classes in sample
        n_support = sample['n_support']         # no. of support images in each class
        n_query = sample['n_query']             # no. of query images in each class

        # seperate the support and query images
        x_support = sample_images[:, :n_support]
        x_query = sample_images[:, n_support:]

        #target indices are 0 ... n_way-1
        target_inds = torch.arange(0, n_way).view(n_way, 1, 1).expand(n_way, n_query, 1).long() # result = torch.Size([n_way, n_query, 1])
        target_inds = Variable(target_inds, requires_grad=False)
        target_inds = target_inds.to(self.device)

        #encode images of the support and the query set
        x = torch.cat([x_support.contiguous().view(n_way * n_support, *x_support.size()[2:]),
                       x_query.contiguous().view(n_way * n_query, *x_query.size()[2:])], 0)

        z = self.encoder.forward(x) # returns embedded vector
        z_dim = z.size(-1) #get the size of the flattenned vector
        # find the mean, that becomes the class prototype
        z_proto = z[:n_way*n_support].view(n_way, n_support, z_dim).mean(1)
        z_query = z[n_way*n_support:]

        #compute distances between vectors of images in the query set and the class prototypes
        dists = euclidean_dist(z_query, z_proto)

        #compute probabilities
        log_p_y = F.log_softmax(-dists, dim=1).view(n_way, n_query, -1)

        loss_val = -log_p_y.gather(2, target_inds).squeeze().view(-1).mean()
        _, y_hat = log_p_y.max(2)
        acc_val = torch.eq(y_hat, target_inds.squeeze()).float().mean()

        # Collect predictions and true labels
        y_true = target_inds.squeeze().cpu().numpy()
        y_pred = y_hat.cpu().numpy()

        return loss_val, {
        'loss': loss_val.item(),
        'acc': acc_val.item(),
        'y_hat': y_hat,
        'y_true' : y_true,
        'y_pred' : y_pred
        }
```

Figure 13: Prototypical Network with Forward Loss Function

```python
# Function to compute the Euclidean distance between feature vectors
def euclidean_dist(x, y):
  """
  Computes euclidean distance btw x and y
  Args:
      x (torch.Tensor): shape (n, d). n usually n_way*n_query
      y (torch.Tensor): shape (m, d). m usually n_way
  Returns:
      torch.Tensor: shape(n, m). For each query, the distances to each centroid
  """
  n = x.size(0)
  m = y.size(0)
  d = x.size(1)
  assert d == y.size(1)

  x = x.unsqueeze(1).expand(n, m, d)
  y = y.unsqueeze(0).expand(n, m, d)

  return torch.pow(x - y, 2).sum(2)
```

Figure 14: Euclidean Distance Function

```python
def test_model_on_one_task(model, n_way, n_support, n_query, test_episodes, x_test, y_test, output_dir, file_name):
    """
    Tests the Prototypical Network on a test set and saves every 100th episode result to a CSV file.

    Args:
        model: trained model
        n_way (int): number of classes in a classification task
        n_support (int): number of images per class in the support set
        n_query (int): number of images per class in the query set
        test_episodes (int): number of episodes to test on
        x_test (np.array): images of testing set
        y_test (np.array): labels of testing set
        output_dir (str): Directory to save the CSV file
        file_name (str): The name of the CSV file to save results

    Returns:
        avg_loss (float): average loss
        avg_acc (float): average accuracy
    """

    running_loss = 0.0
    running_acc = 0.0


    results = []
    model.eval()

    print(f"Saving loss and accuracy every 100 episodes: ")
    with torch.no_grad():
        for episode in range(test_episodes):
            sample = train_and_evaluation.extract_sample(n_way, n_support, n_query, x_test, y_test)
            loss, output = model.set_forward_loss(sample)
            running_loss += output['loss']
            running_acc += output['acc']


            if episode % 100 == 0:
                print(f"Episode: {episode} ---> Loss: {output['loss']:.3f}, Accuracy: {output['acc']:.2f}")

                results.append({
                    'Episode': episode,
                    'Loss': output['loss'],
                    'Accuracy': output['acc']
                })

        avg_loss = running_loss / test_episodes
        avg_acc = running_acc / test_episodes


    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    csv_path = os.path.join(output_dir, file_name)
    df = pd.DataFrame(results)
    df.to_csv(csv_path, index=False)

    return avg_loss, avg_acc
```

Figure 15: Episodic Learning Function

```python
prototypical_model.to(device)
n_way = 2
n_support = 3
n_query = 10

sample_images = train_and_evaluation.extract_sample(n_way, n_support, n_query, x_test, y_test)

results = train_and_evaluation.predict(prototypical_model, sample_images, device)

train_and_evaluation.display_sample(sample_images["images"])

print(f"Class labels sampled: {sample_images['class_labels']}")

print(results)
```

```
Class labels sampled: ['Effusion' 'Covid19']
{'loss': 1.0303239822387695, 'acc': 0.8500000238418579, 'y_hat': tensor([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 0, 0, 1]], device='cuda:0'), 'y_true': array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'y_pred': array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 1]])}
```

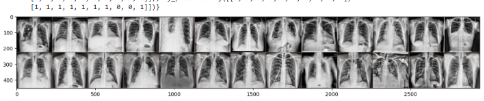


Figure 16: 2 way 3 shot Configuration

Prototypical Network Prediction with resNet50 2 Way 5 Shot Group 1

```python
prototypical_model.to(device)
n_way = 2
n_support = 5
n_query = 15

sample_images = train_and_evaluation.extract_sample(n_way, n_support, n_query, x_test, y_test)

results = train_and_evaluation.predict(prototypical_model, sample_images, device)

train_and_evaluation.display_sample(sample_images["images"])

print(f"Class labels sampled: {sample_images['class_labels']}")

print(results)
```

```
lass labels sampled: ['Covid19' 'Effusion']
'loss': 1.608158826828003, 'acc': 0.8666667342185974, 'y_hat': tensor([[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]], device='cuda:0'), 'y_true': array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'y_pred': array([[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

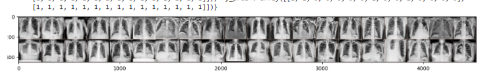


Figure 17: 2 way 5 shot Configuration

```python
def predict(model, sample, device="cpu"):
  """
    Args:
      model (object): trained prototypical model
      sample (dict): dictionary containing the following keys:
                                                    images - images for the support + query set
                                                    n_way - number of classes to sample
                                                    n_support - number of support images
                                                    n_query - number of query images
      device (str): device to run the model on - 'cpu' or 'cuda'

    Returns:
      output (dict): dictionary with the following keys:
                                                loss - loss value
                                                acc - accuracy of prediction
                                                y_hat = prediction tensor for each query image in each class
  """
  model.to(device)
  l, output = model.set_forward_loss(sample)

  return output
```

Figure 18: Predict Function

The Figure below 18 shows the function for calculating classification report.

```python
def generate_classification_report(results, sample_images):
    y_true_flat = results["y_true"].flatten()
    y_pred_flat = results["y_pred"].flatten()


    class_labels = sample_images["class_labels"]
    label_mapping = {i: label for i, label in enumerate(class_labels)}


    report = classification_report(y_true_flat, y_pred_flat, target_names=class_labels)
    return report
```

Figure 19: Code for Classification Report

The Figure 19 below shows the function to plot the confusion matrix

```python
def plot_confusion_matrix(y_true, y_pred, class_labels):
    conf_matrix = confusion_matrix(y_true, y_pred)

    cmap = plt.cm.Blues
    plt.figure(figsize=(8, 6))
    disp = ConfusionMatrixDisplay(conf_matrix, display_labels=class_labels)


    disp.plot(cmap=cmap, values_format='d')


    plt.title("Confusion Matrix", fontsize=12, pad=20)
    plt.grid(False)
    plt.colorbar(disp.im_, fraction=0.046, pad=0.04)


    plt.tight_layout()
    plt.show()
```

Figure 20: Code For Plotting Confusion Matrix

The Figure 20 below shows the function to plot ROC Curve

```python
def plot_roc_curve(results):
    """
    Generates and plots the Receiver Operating Characteristic (ROC) curve.
    """

    y_true_flat = results["y_true"].flatten()
    y_scores = results["y_pred"].flatten()

    fpr, tpr, thresholds = roc_curve(y_true_flat, y_scores, pos_label=1)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.show()
```

Figure 21: Code for Plotting ROC Curve

Figure below shows the code to compare accuracy and loss during episodic learning

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 7))

# Plot Accuracy on the first subplot (ax1)
ax1.plot(df_vgg_2way_3shot['Episode'], df_vgg_2way_3shot['Accuracy'], label='VGG19 Accuracy', color='blue')
ax1.plot(df_resnet_2way_3shot['Episode'], df_resnet_2way_3shot['Accuracy'], label='ResNet50 Accuracy', color='green')
ax1.plot(df_densenet_2way_3shot['Episode'], df_densenet_2way_3shot['Accuracy'], label='DenseNet121 Accuracy', color='red'
ax1.set_xlabel('Episodes')
ax1.set_ylabel('Accuracy')
ax1.set_title('Accuracy vs Episodes (2-Way 3-Shot)')
ax1.legend(loc='upper left')

# Plot Loss on the second subplot (ax2)
ax2.plot(df_vgg_2way_3shot['Episode'], df_vgg_2way_3shot['Loss'], label='VGG19 Loss', color='blue', linestyle='--')
ax2.plot(df_resnet_2way_3shot['Episode'], df_resnet_2way_3shot['Loss'], label='ResNet50 Loss', color='green', linestyle='
ax2.plot(df_densenet_2way_3shot['Episode'], df_densenet_2way_3shot['Loss'], label='DenseNet121 Loss', color='red', linest
ax2.set_xlabel('Episodes')
ax2.set_ylabel('Loss')
ax2.set_title('Loss vs Episodes (2-Way 3-Shot)')
ax2.legend(loc='upper left')

# Adjust the layout to make sure everything fits
plt.tight_layout()

# Show the plot
plt.show()
```

Figure 22: Code for Comparing Accuracy and Loss

```python
def plot_metrics_from_csv(*csv_files, figsize=(12, 7)):
    """
    Plots precision, recall, and F1 score for multiple models from CSV files.

    :param csv_files: Variable length argument list of CSV file paths.
    :param figsize: Tuple specifying the size of the figure, default is (12, 7).
    :return: None, displays the plot.
    """
    metrics = ['Precision', 'Recall', 'F1 Score']
    data = {metric: [] for metric in metrics}
    models = []

    for csv_file in csv_files:
        model_name = csv_file.split('/')[-2]  # Assuming model name is in the directory name
        models.append(model_name)
        df = pd.read_csv(csv_file)

        def compute_metrics(df):
            y_true = df['True Class'].values
            y_pred = df['Pred Class'].values
            return (
                precision_score(y_true, y_pred, average='weighted'),
                recall_score(y_true, y_pred, average='weighted'),
                f1_score(y_true, y_pred, average='weighted')
            )

        precision, recall, f1 = compute_metrics(df)
        data['Precision'].append(precision)
        data['Recall'].append(recall)
        data['F1 Score'].append(f1)

    # Create the plot
    fig, ax = plt.subplots(figsize=figsize)

    x = range(len(models))
    width = 0.25

    for i, metric in enumerate(metrics):
        offset = width * i
        ax.bar([x + offset for x in range(len(models))], data[metric], width,
               label=metric)

    ax.set_ylabel('Score')
    ax.set_title('Comparison of Precision, Recall, and F1 Score Across Models')
    ax.set_xticks([r + width for r in range(len(models))])
    ax.set_xticklabels(models)
    ax.legend()

    plt.tight_layout()
    plt.show()
```

Figure 23: Function to compare metrics of different models