National College of Ireland

Configuration Manual

MSc Research Project
Masters of Science in Artificial Intelligence

Ivan Rodriguez
Student ID: X23430133

School of Computing
National College of Ireland

Supervisor: Arundev Vamadevan

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ivan Rodriguez
Student ID:	X23430133
Programme:	Masters of Science in Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Arundev Vamadevan
Submission Due Date:	11/08/2025
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:	
Date:	10th August 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

Ivan Rodriguez
X23430133

1 Introduction

This is the configuration manual for "Exploiting Explainable AI tools for improving model classification explainability". It contains the basics for executing the related code artifact. Should any problem arise feel free to mail x23430133@student.ncirl.ie where clarification will be provided at the shortest possible notice.

2 System Specifications

Figure 1: System Specifications

The computer used to process the code artifact was a Macbook Pro (15in 2018) with a 2.2 GHz 6-Core Intel Core i7 processor.

2.1 Software Used

Used default terminal with Brew, Python, and Xcode tools. For code creation and manipulation used VSCode.

- Terminal

  - Xcode
  - Brew
  - Python
    - Tensorflow
    - Lime
    - Shap
    - GradCam
    - scikit
- VSCode

3 Dataset Specifications

This study used 3 different datasets, all related to healthcare classification images. The full dataset folder hierarchy includes all 3 models as well as space for the individual datasets. Each dataset is divided by train and test and validation data and each of the internal folders is subdivided between categories, either positive or negative. Initial data folder level might vary according to each dataset but needs to be selected properly within the code (default names should work properly).

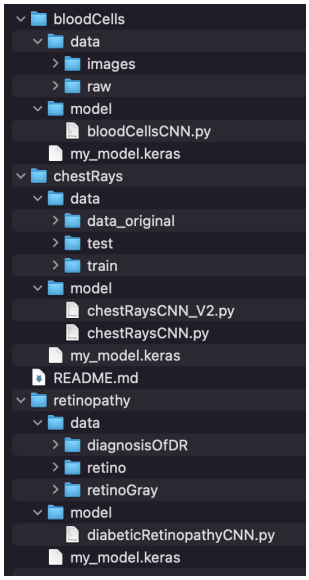

Figure 2: File structure

4 Libraries

- Xcode
- Brew
- Python

- numpy
- Tensorflow
- Lime
- Shap
- GradCam
- scikit

5 Executing Code

Assuming Library section software is installed, next steps include installing Python libraries and running the code. The use of Virtual Environments is encouraged but will not be explained in the scope of this manual.

In the spirit of keeping the manual short only one of the 3 models will be explained in detail but the other 2 are similar enough such that the biggest difference should be

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import shap
import lime
import lime.lime_image
from skimage.segmentation import mark_boundaries

DATA_DIR = './chestRays/data/'
TRAIN_DIR = os.path.join(DATA_DIR, 'train')
TEST_DIR = os.path.join(DATA_DIR, 'test')

IMG_WIDTH = 200
IMG_HEIGHT = 200
IMG_CHANNELS = 3
IMG_SHAPE = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)
BATCH_SIZE = 64
EPOCHS = 30
LEARNING_RATE = 0.00001
CLASS_NAMES = ['NORMAL', 'COVID19', 'PNEUMONIA']
NUM_CLASSES = len(CLASS_NAMES)
```

Figure 3: Library Setup

```python
train_dataset = tf.keras.utils.image_dataset_from_directory(
    TRAIN_DIR,
    labels='inferred',
    label_mode='categorical',
    class_names=CLASS_NAMES,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_split=0.15,
    subset='training',
    seed=123
)

validation_dataset = tf.keras.utils.image_dataset_from_directory(
    TRAIN_DIR,
    labels='inferred',
    label_mode='categorical',
    class_names=CLASS_NAMES,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_split=0.15,
    subset='validation',
    seed=123
)

test_dataset = tf.keras.utils.image_dataset_from_directory(
    TEST_DIR,
    labels='inferred',
    label_mode='categorical',
    class_names=CLASS_NAMES,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=False
)

def normalize(image, label):
    image = tf.cast(image, tf.float32) / 255.0
    return image, label

train_dataset = train_dataset.map(normalize)
validation_dataset = validation_dataset.map(normalize)
test_dataset = test_dataset.map(normalize)
```

Figure 4: Dataset and variable setup

```python
def create_cnn_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    # Block 1
    x = Conv2D(filters=3, kernel_size=(3, 3), activation='relu')(inputs)
    x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.05)(x)

    # Block 2
    x = Conv2D(filters=96, kernel_size=(3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(3, 3))(x)
    x = Dropout(0.2)(x)

    # Block 3
    x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.1)(x)

    # Block 4
    x = Conv2D(filters=256, kernel_size=(3, 3), activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.1)(x)

    # Flattening and Dense Layers
    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.45)(x)

    # Output Layer
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model

> if(not os.path.exists('my_model.keras')):…
> else:…
```

Figure 5: CNN Model

```
if(not os.path.exists('my_model.keras')):
    model = create_cnn_model(IMG_SHAPE, NUM_CLASSES)

    model.compile(optimizer=Adam(learning_rate=LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    print("\n--- Starting Model Training ---")
    history = model.fit(
        train_dataset,
        epochs=EPOCHS,
        validation_data=validation_dataset,
        callbacks=[early_stopping]
    )
    print("\nEvaluating the Proposed Model...")

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(EPOCHS)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.savefig("training_validation_loss.png")
    model.save('my_model.keras')
else:
    model = tf.keras.models.load_model('my_model.keraspreviously')
```

Figure 6: Load or Train Model

```python
# Extract a batch from the test dataset for explanation
# Get the first image in the batch
test_images, test_labels = next(iter(test_dataset))
sample_image = test_images[0:1]
sample_label = test_labels[0]

background_images, _ = next(iter(train_dataset.take(1)))

print(f"\n--- Generating XAI Explanations for a sample image ---")
print(f"True Label: {CLASS_NAMES[np.argmax(sample_label)]}")
preds = model.predict(sample_image)
print(f"Predicted Label: {CLASS_NAMES[np.argmax(preds)]} with probability {np.max(preds):.4f}")


# SHAP
explainer_shap = shap.GradientExplainer(model, background_images.numpy())
shap_values = explainer_shap.shap_values(sample_image.numpy())

print("\nDisplaying SHAP explanation plot...")
shap.image_plot(shap_values, -sample_image.numpy(), show=False)
plt.suptitle("SHAP Explanation")
plt.savefig('shap01.png')

# LIME
explainer_lime = lime.lime_image.LimeImageExplainer()
explanation_lime = explainer_lime.explain_instance(
    sample_image[0].numpy().astype('double'),
    model.predict,
    top_labels=1,
    hide_color=0,
    num_samples=1000
)

print("\nDisplaying LIME explanation plot...")
temp, mask = explanation_lime.get_image_and_mask(
    explanation_lime.top_labels[0],
    positive_only=True,
    num_features=5,
    hide_rest=False
)

plt.imshow(mark_boundaries(temp, mask))
plt.title("LIME Explanation")
plt.axis('off')
plt.savefig('lime01.png')
```

Figure 7: Shap and Lime

```python
# Grad-CAM
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = tf.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()

# Find the name of the last convolutional layer
last_conv_layer_name = [layer.name for layer in model.layers if "conv2d" in layer.name][-1]
heatmap = make_gradcam_heatmap(sample_image, model, last_conv_layer_name)

print("\nDisplaying Grad-CAM heatmap...")
plt.matshow(heatmap)
plt.title("Grad-CAM Heatmap")
plt.savefig('gradcam01.png')

# Superimpose the heatmap on the original image
img = sample_image[0].numpy()
heatmap = np.uint8(255 * heatmap)
jet = plt.get_cmap("jet")
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]
jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)

superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = tf.keras.preprocessing.image.array_to_img(superimposed_img)

plt.imshow(superimposed_img)
plt.title("Grad-CAM Superimposed")
plt.axis('off')
plt.savefig('gradcam_superimposed01.png')
```

Figure 8: Grad-cam