National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Pranav Prakash
Student ID: x23134681

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pranav Prakash
Student ID:	X23134682
Programme:	MSc. Data Analytics **Year:** Jan2024 – Jan2025
Module:	MSc. Research Project
Lecturer:	Furqan Rustam
Submission Due Date:	December 12, 2024
Project Title:	Severity Classification of Knee Osteoarthritis from X-Ray Images using Deep Learning
Word Count:	520 **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: Pranav Prakash

Date: December 12, 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pranav Prakash
x23134682

1 Introduction

This is a step-by-step manual on setting up the code and running those for classifying knee osteoarthritis images coloured severity levels. The paper describes the hardware required, the dependencies, the data acquisition, and the execution procedures.

2 Hardware and Software Specification

This section contains hardware and software set up to the project using powerful cloud-based resources provided by the Google Colab Pro.

Category	**Specification**
Platform	Google Colab Pro
Processor	NVIDIA A100 Tensor Core GPU
System RAM	Up to 83.5 GB
GPU RAM	40.0 GB
Storage	Up to 235.7 GB available in Colab runtime
Operating System	Cloud-based (Google Colab environment)
System Type	Cloud-based, x64 architecture
Programming Language	Python 3.8 or higher
IDE	Google Colab Pro interface with Jupyter Notebook compatibility

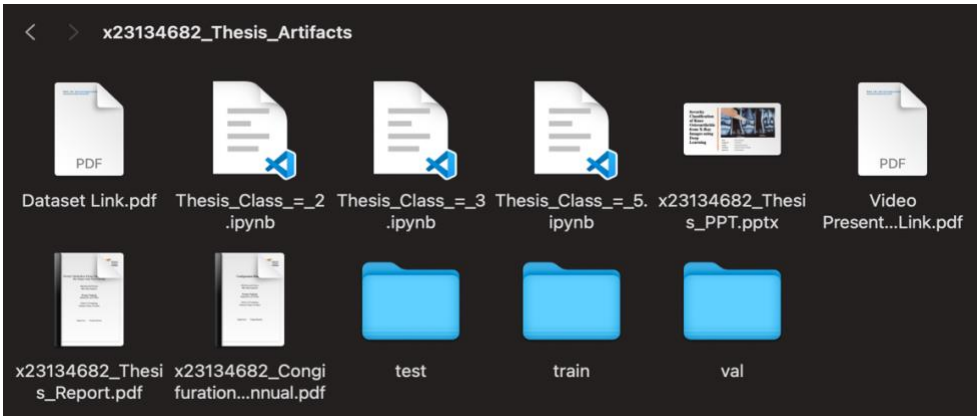
The python packages used are as follows:

Library	Usage
TensorFlow	Building and training deep learning models
NumPy	Numerical computations and array manipulation
OpenCV	Image pre-processing (e.g., resizing, grayscale conversion)
Matplotlib	Data visualization (e.g., plotting loss and accuracy curves)
Seaborn	Advanced data visualization (e.g., confusion matrix heatmaps)
Pandas	Data manipulation and tabular data handling
Scikit-learn	Model performance evaluation metrics

Use the following script to check for the existence of the main libraries and install them if missing:

```
import importlib
import subprocess

# List of required libraries
required_libraries = {
        "tensorflow": "tensorflow",
        "numpy": "numpy",
        "opencv-python": "cv2",
        "matplotlib": "matplotlib",
        "seaborn": "seaborn",
        "pandas": "pandas",
        "scikit-learn": "sklearn"
}

# Install missing libraries
for library, module in required_libraries.items():
        try:
                Importlib.import_module(module)
        except ImportError:
                print(f"{module} not found. Installing {library}...")
                subprocess.check_call(["pip", "install", library])
```

3 Artifacts

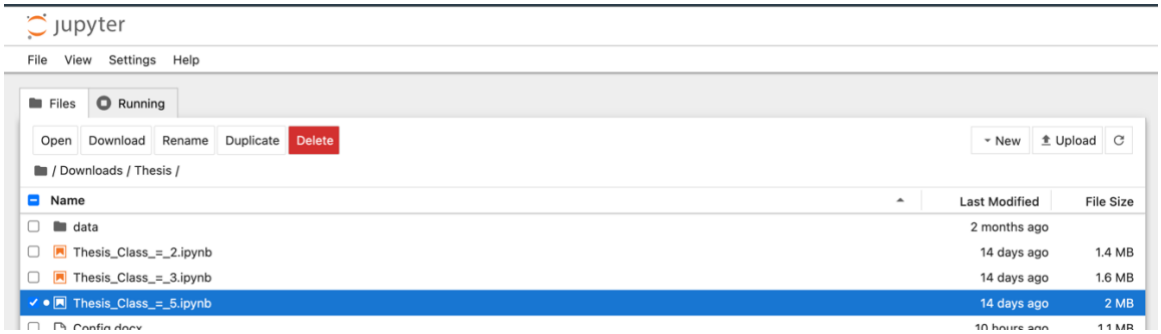


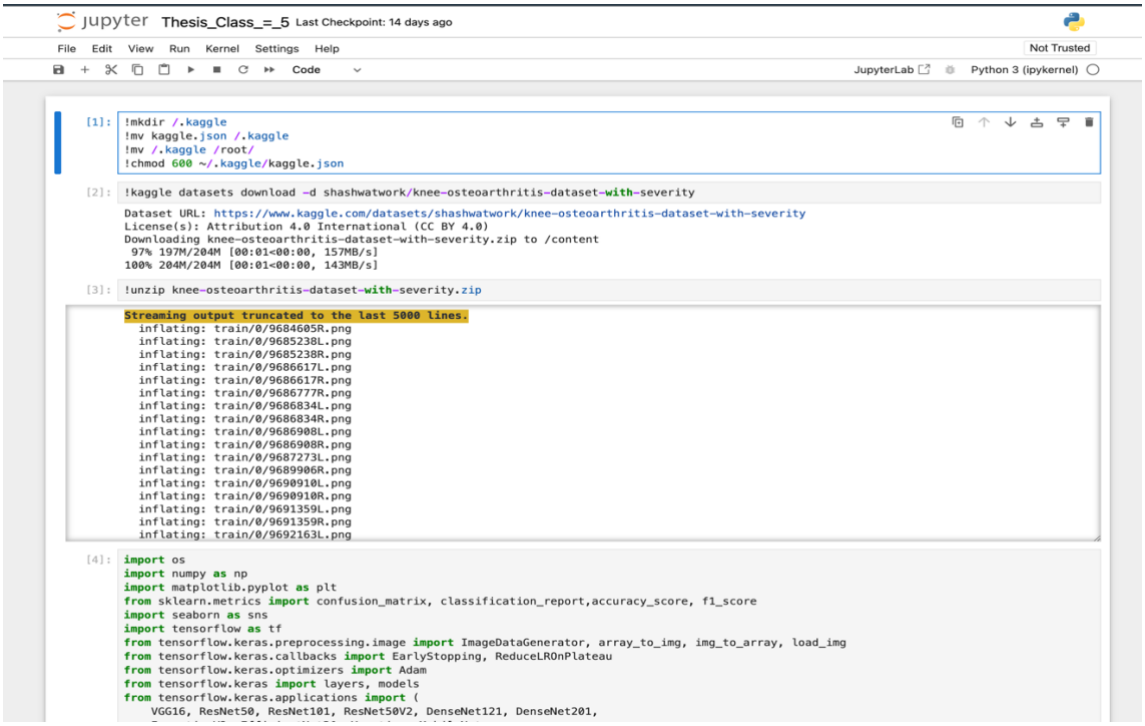
- Dataset Link.pdf - Contains the link to the Kaggle dataset.
- Thesis_Class_=_2.ipynb – Code for 2 class case study.
- Thesis_Class_=_3.ipynb – Code for 3 class case study.
- Thesis_Class_=_5.ipynb – Code for 5 class case study.
- x23134682_Thesis_PPT.pptx – Thesis presentation.
- Video Presentation Link.pdf – Link to presentation video.
- x23134682_Configuration_Mannual.pdf
- x23134682_Thesis_Report.pdf
- test, train, val - Data

4 Implementation

4.1 Opening Jupyter Notebook

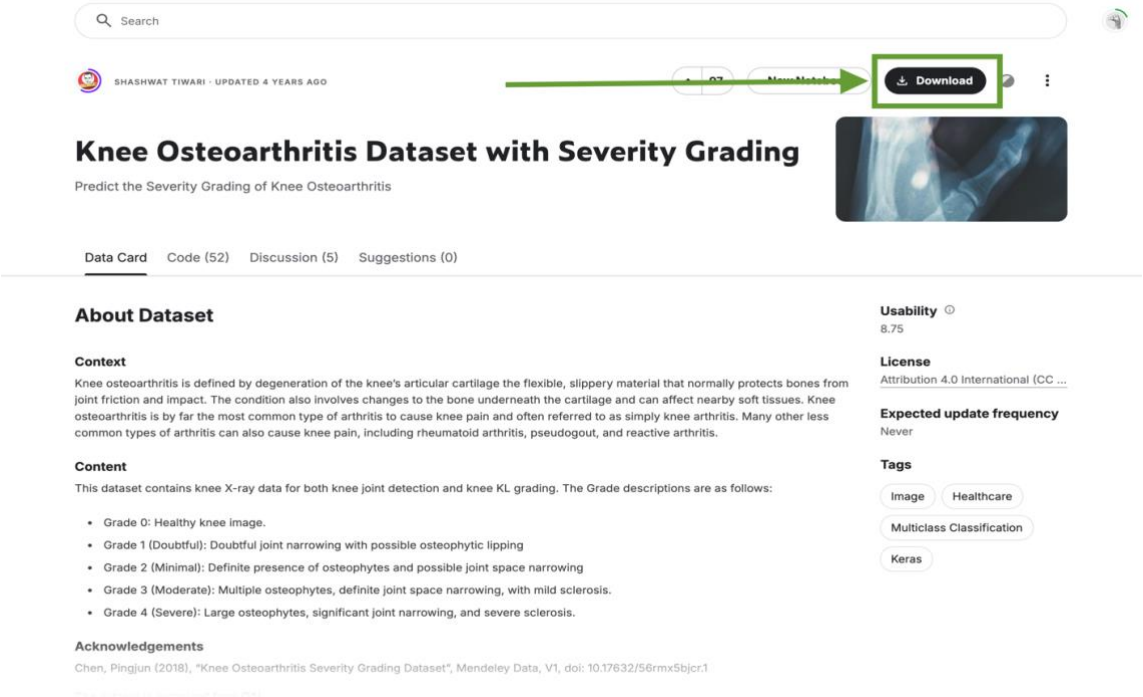
Open the required python notebook based on the case study (class 5,3 or 2) on jupyter.





4.2 Dataset

The dataset used in this project is the **Knee Osteoarthritis Dataset with Severity**, which can be downloaded from Kaggle using the following link:
https://www.kaggle.com/datasets/shashwatwork/knee-osteoarthritis-dataset-with-severity



4.3 Granularity Conversion

Converting the dataset into the required classes based on the case study. The below function "reorganize_folder_structure()" is used to reduce the classes from 5 to 3 or 2.

```
base_dirs = ['train', 'test', 'val']

def reorganize_folder_structure(base_dir):
    for class_dir in ['0','1','2','3','4']:
        class_path = os.path.join(base_dir, class_dir)
        if os.path.isdir(class_path):
            new_class = class_mapping[class_dir]
            new_class_path = os.path.join(base_dir, new_class)
            if not os.path.exists(new_class_path):
                os.makedirs(new_class_path)

            for img_file in os.listdir(class_path):
                src_path = os.path.join(class_path, img_file)
                dst_path = os.path.join(new_class_path, img_file)
                shutil.move(src_path, dst_path)

            shutil.rmtree(class_path)


for base_dir in base_dirs:
    reorganize_folder_structure(base_dir)

print("Folder reorganization completed.")
```

```
Folder reorganization completed.
```

Only the dictionary "class_mapping" varies for different case studies.

Case Study	class_mapping
Class = 5	NIL
Class = 3	`class_mapping = { '0': '0', '1': '0', '2': '1', '3': '1', '4': '2' }`

Class = 2	`class_mapping = { '0': '0', '1': '0', '2': '1', '3': '1', '4': '1' }`

4.4 Image Pre-processing

Preparing the image dataset before training CNN models:

```
[ ] # Image dimensions and batch size
    IMG_HEIGHT = 224
    IMG_WIDTH = 224
    BATCH_SIZE = 64
```

```
[ ] # Custom preprocessing function to convert grayscale to 3-channel grayscale
    def preprocess(image):
        grayscale = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        resized = cv2.resize(grayscale, (IMG_HEIGHT, IMG_WIDTH))
        bw_3channel = cv2.merge([resized, resized, resized])
        return bw_3channel

    # Normalization
    train_datagen = ImageDataGenerator(rescale=1.0 / 255, preprocessing_function=preprocess)
    val_datagen = ImageDataGenerator(rescale=1.0 / 255, preprocessing_function=preprocess)
    test_datagen = ImageDataGenerator(rescale=1.0 / 255, preprocessing_function=preprocess)
```

```
    # Data Generators
    train_generator = train_datagen.flow_from_directory(
        train_dir, target_size=(IMG_HEIGHT, IMG_WIDTH), batch_size=BATCH_SIZE, class_mode="categorical",
    )
    val_generator = val_datagen.flow_from_directory(
        val_dir, target_size=(IMG_HEIGHT, IMG_WIDTH), batch_size=BATCH_SIZE, class_mode="categorical",
    )
    test_generator = test_datagen.flow_from_directory(
        test_dir, target_size=(IMG_HEIGHT, IMG_WIDTH), batch_size=BATCH_SIZE, class_mode="categorical", shuffle=False
    )
```

Preparing the image dataset before training traditional ML models:

```
def preprocess_images(directory, target_size=(224, 224)):
    images = []
    labels = []
    class_labels = os.listdir(directory)
    class_map = {label: idx for idx, label in enumerate(class_labels)}

    for label in class_labels:
        label_dir = os.path.join(directory, label)
        for file in os.listdir(label_dir):
            img_path = os.path.join(label_dir, file)
            img = load_img(img_path, target_size=target_size)
            img_array = img_to_array(img) / 255.0
            images.append(img_array.flatten())
            labels.append(class_map[label])

    return np.array(images), np.array(labels), class_map
```

```
[ ] train_images, train_labels, class_map = preprocess_images(train_dir)
    test_images, test_labels, _ = preprocess_images(test_dir)
```

```
X_train, X_val, y_train, y_val = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)
```

4.5 Modelling

4.5.1 Deep Learning Models

```python
# Function to build, train, and evaluate a model
def train_and_evaluate_model(base_model, model_name):
    print(f"\nTraining {model_name}...\n")
    # Load the base model
    base_model = base_model(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3), include_top=False, weights="imagenet")

    # Add custom layers on top
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation="relu"),
        layers.Dropout(0.5),
        layers.Dense(train_generator.num_classes, activation="softmax")
    ])

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.001), loss="categorical_crossentropy", metrics=["accuracy"])

    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-20, verbose=1)
    class_weights = compute_class_weight(
        class_weight="balanced",
        classes=np.unique(train_generator.classes),
        y=train_generator.classes
    )
    class_weights = dict(enumerate(class_weights))

    # Train the model
    history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs=20,
        batch_size=BATCH_SIZE,
        class_weight=class_weights,
        callbacks=[reduce_lr]
    )

    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(test_generator)
    print(f"\nTest Accuracy for {model_name}: {test_accuracy * 100:.2f}%\n")

    # Predict on the test set
    predictions = model.predict(test_generator)
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = test_generator.classes
    class_labels = list(test_generator.class_indices.keys())
```

The “train_and_evaluate()” function is used to train the CNN models by sending base_model and model_name as parameters.

```python
# DenseNet121
train_and_evaluate_model(DenseNet121, "DenseNet121")
```

```
Training DenseNet121...

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 313s 2s/step - accuracy: 0.2569 - loss: 1.6963 - val_accuracy: 0.3971 - val_loss: 5.3429 - learning_rate: 0.0010
Epoch 2/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.4202 - loss: 1.1362 - val_accuracy: 0.3971 - val_loss: 5.5910 - learning_rate: 0.0010
Epoch 3/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.4454 - loss: 1.0693 - val_accuracy: 0.4492 - val_loss: 1.3587 - learning_rate: 0.0010
Epoch 4/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 146ms/step - accuracy: 0.5053 - loss: 0.9708 - val_accuracy: 0.3971 - val_loss: 6.2531 - learning_rate: 0.0010
Epoch 5/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.4575 - loss: 1.0281 - val_accuracy: 0.4322 - val_loss: 3.0852 - learning_rate: 0.0010
Epoch 6/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 145ms/step - accuracy: 0.4977 - loss: 0.9668 - val_accuracy: 0.4722 - val_loss: 1.3268 - learning_rate: 0.0010
Epoch 7/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.5288 - loss: 0.9159 - val_accuracy: 0.4939 - val_loss: 1.2497 - learning_rate: 0.0010
Epoch 8/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.5501 - loss: 0.8701 - val_accuracy: 0.5557 - val_loss: 1.0548 - learning_rate: 0.0010
```

Above is an example of running DenseNet121 model using the “train_and_evaluate()” function.

4.5.2 Machine Learning Models

```python
models = {
    "RandomForest": RandomForestClassifier(n_estimators=100, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "NaiveBayes": GaussianNB()
}

for model_name, model in models.items():
    print(f"\nTraining {model_name}...")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
```

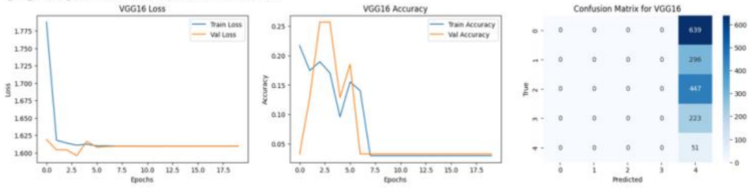
The above code is used to train ML models – Random Forest, KNN & Naïve Bayes

4.6 Evaluation

CNN models metrics:

```python
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"\nTest Accuracy for {model_name}: {test_accuracy * 100:.2f}%\n")

# Predict on the test set
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification report
print(f"Classification Report for {model_name}:\n")
print(classification_report(true_classes, predicted_classes, target_names=class_labels))

test_accuracies[model_name] = test_accuracy * 100

# Plot training history
plt.figure(figsize=(20, 4))

# Loss Plot
plt.subplot(1, 3, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title(f'{model_name} Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Accuracy Plot
plt.subplot(1, 3, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title(f'{model_name} Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)
plt.subplot(1, 3, 3)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.title(f"Confusion Matrix for {model_name}")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

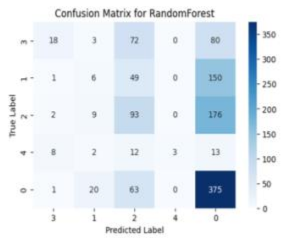
```
Test Accuracy for VGG16: 3.08%

26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 155ms/step
Classification Report for VGG16:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       639
           1       0.00      0.00      0.00       296
           2       0.00      0.00      0.00       447
           3       0.00      0.00      0.00       223
           4       0.03      1.00      0.06        51

    accuracy                           0.03      1656
   macro avg       0.01      0.20      0.01      1656
weighted avg       0.00      0.03      0.00      1656

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



ML Model metrics:

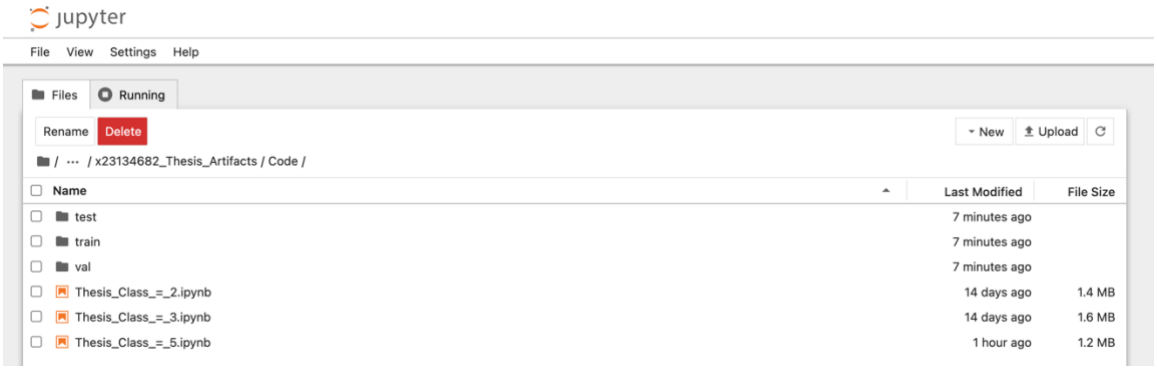
```python
# Calculate accuracy
accuracy = accuracy_score(y_val, y_pred)
print(f"{model_name} Accuracy: {accuracy * 100:.2f}%")
test_accuracies[model_name] = accuracy * 100

# Classification report
print(f"Classification Report for {model_name}:\n")
print(classification_report(y_val, y_pred, target_names=list(class_map.keys())))

# Confusion Matrix
cm = confusion_matrix(y_val, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=list(class_map.keys()), yticklabels=list(class_map.keys()))
plt.title(f"Confusion Matrix for {model_name}")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Training RandomForest...
RandomForest Accuracy: 42.82%
Classification Report for RandomForest:

              precision    recall  f1-score   support

           3       0.60      0.10      0.18       173
           1       0.15      0.03      0.05       206
           2       0.32      0.33      0.33       280
           4       1.00      0.08      0.15        38
           0       0.47      0.82      0.60       459

    accuracy                           0.43      1156
   macro avg       0.51      0.27      0.26      1156
weighted avg       0.41      0.43      0.36      1156
```



5 Executing the code

- Download and unzip data.
- Copy train, test and val folders to the same directory as the python notebooks.
- Make sure the code artifacts are in similar structure down below.



- Open each jupyter notebook.
- In the task bar, click Run -> Run All Cells.