Configuration Manual

MSc Research Project
Data Analytics

Karen Hernandez Abasolo
Student ID: X20118210

School of Computing
National College of Ireland

Supervisor: Dr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Karen Hernandez Abasolo
Student ID:	X20118210
Programme:	Data Analytics **Year:** 2020-2021
Module:	MSc Research Project
Lecturer:	Dr. Hicham Rifai
Submission Due Date:	16/08/2021
Project Title:	Detection of Knee Osteoarthritis Severity using a Fusion of Machine and Deep Learning models
Word Count:	1931 **Page Count:** 18

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: Karen Hernandez Abasolo

Date: 16/08/2020

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

Karen Hernandez Abasolo
Student ID: x20118210

1 Introduction

The current manual configuration aims to replicate the proposed research project from scratch. It contains hardware and software requirements, also all the packages, libraries, and programming codes performed during each stage of the implementation.

2 System Configurations

2.1 Hardware

Operating System: Windows 10
Processor: Intel(R) Core (TM) i5-6300U CPU @ 2.40GHz 2.50 GHz
Installed RAM: 8.00 GB (7.88 GB usable)

2.2 Software

The following software enabled the implementation:

Microsoft Office: Excel
Anaconda Navigator for Windows (Version 1.9.7)
Jupyter Notebook (Version 6.3.0)
Python (Version 3.8.8)

2.2.1 Python Environment Setup

Machine learning models were completely implemented on Jupyter Notebook hosted by Anaconda framework, using python language. The last stage of the project which consisted of a fusion of machine and deep learning models was implemented in this environment.

2.2.2 Google Colab Environment Setup

Deep learning models were implemented on Google Colaboratory, a product from Google Research that allows to write and execute code in Python language. GPU was set as a hardware accelerator.

3 Project Implementation

The current research project involves three main stages: implementation of machine learning models, deep learning models, and a fusion system that combines both predictions provided by them. For better understanding, the manual configuration will explain all stages of each process.

Machine Learning Models

3.1.1 Data Gathering

The first step is getting the dataset from OAI study[1]. It is required to create a user and login into the account as shown in Figure 1.

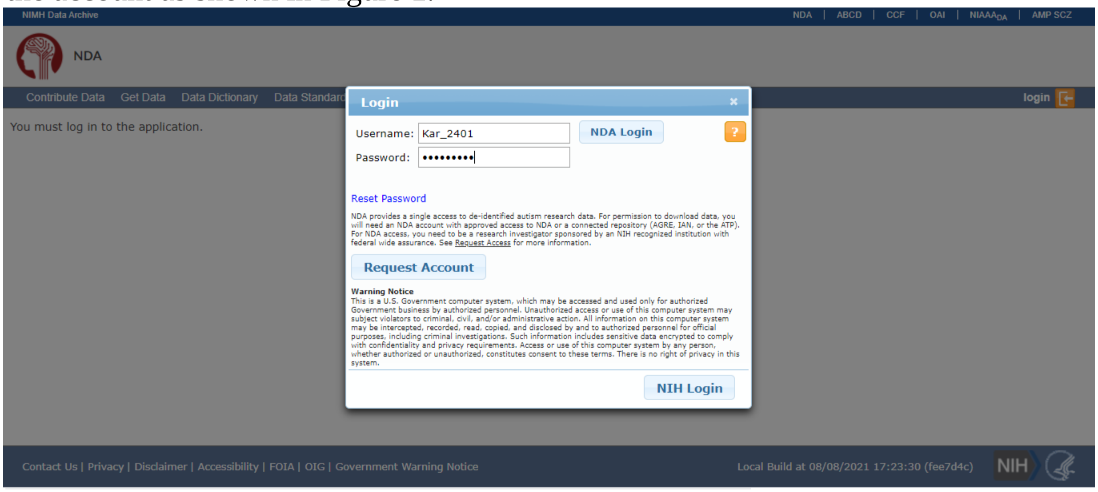

Figure 1. Login to OAI Study

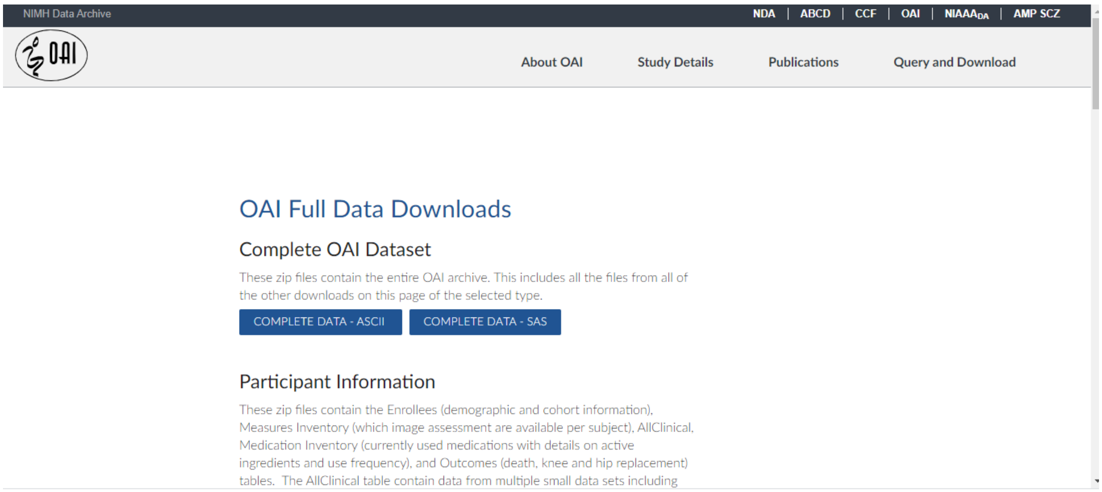

Figure 2. Clinical Dataset from OAI study

Clinical data is downloaded from the website. It is a zipped file that contains clinical data of each visit of the patients. The current project only considered the file "AllClinical00" which is data at the baseline of the study.

[1] The Osteoarthritis Initiative: https://nda.nih.gov/oai

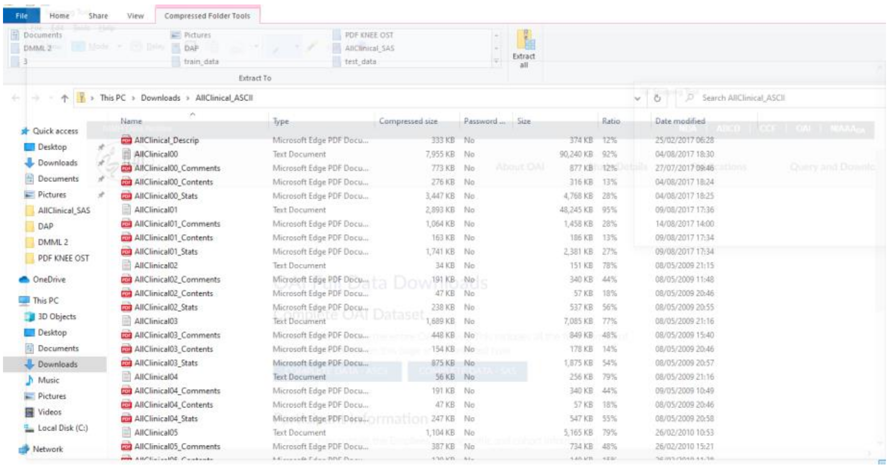

Figure 3. Dataset Files

There are important attributes such as Sex and Race that were merged from another file as is shown in Figure 4.

```
#Reading files
df_clinical = pd.read_csv("C:/Users/karen/Downloads/Clinical00.csv")
df_measinventory = pd.read_csv("C:/Users/karen/Downloads/measinventory.csv")
```

```
#Adding variables from another dataset
df_koa=pd.merge(df_clinical,df_measinventory[["ID","P02SEX","P02RACE"]],how="left")
```

Figure 4. Accessing clinical data from the system

3.1.2 Data Preparation

Libraries required to explore, impute missing values, graph plots, and statistical analysis of clinical data are shown in Figure 5.

```
#Libraries required to preprocess clinical data
import pandas as pd
import numpy as np

import category_encoders as ce

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
```

Fig 5. Libraries required to preprocess clinical data

Before starting the preprocessing stage, all dataframe was duplicated and a single ID was created because the variable outcome is for each knee of the patient, however, clinical data is a single row per patient, that process is illustrated in Figure 6.

```
ID_L=df_koafinal["ID"].astype(str)+"L"
df_koafinalL=df_koafinal.assign(ID_SIDE=ID_L.values)
df_koafinalL
```

```
ID_R=df_koafinal["ID"].astype(str)+"R"
df_koafinalR=df_koafinal.assign(ID_SIDE=ID_R.values)
df_koafinalR
```

Fig 6. Creating a unique ID for both knees of a patient

A column with more than 50% of missing values is dropped. Besides, due to the sensitivity of patient information, all rows with NA values are dropped.

```
df_koa_clean1=df_koa.drop(columns=["V00RAMEDS","V00KOOSFX3","P01KPNLEVY","V00SMKNOW","V00KOOSFX2","V00FFQYR82","V00HOURWK","V00K(
df_koa_clean1.head()
```

```
df_koa_final=df_koa_clean3.dropna()
df_koa_final
```

Fig 7. Deleting missing values

The target variable comes from x-ray images, and it is needed to merge it with clinical dataset, this process is in Fig 8.

```
#Merging target variable
```

```
df_target = pd.read_excel("C:/Users/karen/Desktop/kaggle_OA/target_variable.xlsx")
```

```
df_koa_v2=pd.merge(df_koa_v1,
                   df_target,
                  how="left")
df_koa_v2
```

Fig 8. Merging target variable with clinical dataframe

```
filter1=df_koa_v3['V00ABCIRC']>138
filtered_df = df_koa_v3[filter1]
filtered_df
```

```
filter1=df_koa_v3['V00ABCIRC']>138
filtered_df = df_koa_v3[filter1]
filtered_df
```

Fig 9. Deleting outliers in the dataframe

```
from sklearn.preprocessing import MinMaxScaler

column_names_to_not_normalize = ['ID', 'ID_SIDE', 'KLGRADE']
column_to_normalize1 = [x for x in list(transf_data) if x not in column_names_to_not_normalize ]

z = transf_data[column_to_normalize1].values
scaler = MinMaxScaler()
z_scaled = scaler.fit_transform(z)
df_tempo = pd.DataFrame(z_scaled, columns=column_to_normalize1, index = transf_data.index)
transf_data[column_to_normalize1] = df_tempo
df_tempo
```

Fig 10. Normalization of variables

3.1.3 Modelling machine learning methods

Implementation of machine learning requires a set of libraries to be set described in Figure 11.

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from imblearn.pipeline import make_pipeline as make_pipeline_imb # To do our transformation in a unique time
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import make_pipeline
from imblearn.metrics import classification_report_imbalanced

#Library to split the dataset
from sklearn.model_selection import train_test_split
from collections import Counter

from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.ensemble import GradientBoostingClassifier


#Libraries for tunning the model
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score, GridSearchCV

#Libraries for evaluation
from scikitplot.metrics import plot_roc
from scikitplot.metrics import plot_precision_recall
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score, recall_score, fbeta_score, confusion_matrix
from sklearn.metrics import precision_recall_curve, accuracy_score, classification_report
```

Figure 11. Libraries required to implement machine learning models

SPLIT INTO TRAIN AND TEST

```python
X = koa_hotencoded_allvar.drop(["KLGRADE"], axis=1) #Setting the X to do the split
X.set_index('ID_SIDE', inplace=True)
X.values
y = koa_hotencoded_allvar["KLGRADE"].values # transforming the values in array
```

```python
X_col = koa_hotencoded_allvar.drop(["KLGRADE","ID_SIDE"], axis=1)
```

```python
# splitting data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2, test_size=0.30)
```

```python
#Consider train and test data to divide the same groups in image models
X_train_df = pd.DataFrame(X_train)
X_test_df = pd.DataFrame(X_test)
```

```python
X_train_df.to_csv("C:/Users/karen/Desktop/train_data/train2.csv")
X_test_df.to_csv("C:/Users/karen/Desktop/test_data/test2.csv")
```

Figure 12. Train and Test datasets are created

To overcome dealing with an imbalanced dataset, SMOTE technique is applied to enhance the models, the implementation of the technique is shown in Figure 13.

```python
#SMOTE will oversample all classes to have the same number of examples as the class with the most examples.
# transform the dataset
oversample = SMOTE()
X_train_SMOTE, y_train_SMOTE = oversample.fit_resample(X_train_final, y_train)
# summarize distribution
counter = Counter(y_train_SMOTE)
for k,v in counter.items():
    per = v / len(y_train_SMOTE) * 100
    print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
plt.bar(counter.keys(), counter.values())
plt.show()
```

Figure 13. SMOTE strategy applied in Train dataset

Random Forest

Figure 14 shows the implementation of Random Forest, this is the first machine learning model. A search of the best parameters to improve it is conducted by GridSearchCV. Three experiments were implemented here, Random Forest with hyperparameters, Random Forest taking into consideration hyperparameter and SMOTE technique, and Random Forest with hyperparameters and Weighted dataset. In the end, the importance of features in this model is plotted.

```
rf = RandomForestClassifier(max_depth=4, n_estimators=20)
rf.fit(X_train_final, y_train)

# Run prediction on test set.
y_pred_rf = rf.predict(X_test_final)
y_pred_train_rf=rf.predict(X_train_final)

#Evaluating
print("Train accuracy::",accuracy_score(y_train,y_pred_train_rf))
print("Test accuracy::",accuracy_score(y_test,y_pred_rf))
```

Figure 14. Implementation of Random Forest, a first model with parameters set by default

```
#Tunning parameters
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [20,100, 200, 300, 1000]}

rf_grid = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)

rf_grid.fit(X_train_final,y_train)
rf_grid.cv_results_, rf_grid.best_params_,rf_grid.best_score_
```

Fig 15. Tunning parameters by GridSearchCV with 3 cross-validation in the process.

```
rf_weig = RandomForestClassifier(max_depth=90, max_features=2,min_samples_leaf=3, min_samples_split=8,
                                 n_estimators=300,class_weight='balanced')

rf_weig.fit(X_train_final, y_train)

k_fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

new_scores = cross_val_score(rf_weig, X_train_final, y_train, cv=k_fold, n_jobs=-1)
print("Cross validation train accuracy score:", new_scores.mean())
new_scores_test = cross_val_score(rf_weig, X_test_final, y_test, scoring='accuracy', cv=k_fold, n_jobs=-1)
print("Cross validation train accuracy score:", new_scores_test.mean())
```

Fig 16. Random Forest version after applying hyper parametrization, implementing option that balance the dataset.

```
plt.title("Feature Importance",fontsize=25)
plt.bar(range(Xtrain_df.shape[1]),importances[sorted_indices],align="center")
plt.xticks(range(Xtrain_df.shape[1]),Xtrain_df.columns[sorted_indices],rotation=90)
plt.tight_layout()
#plt.figure(figsize=(8, 6))
#plt.xlabel('Feature importance score', fontsize=20)
plt.show()
```

Fig 17. Plotting Feature Importance in Random Forest Model

Gradient Boosting

A baseline Gradient Boosting model is implemented however, to enhance it is applied a set of searches of the best parameters as shown below.

```
#Tunning parameters
p_test3 = {'learning_rate':[0.15,0.1,0.05,0.01,0.005,0.001], 'n_estimators':[100,250,500,750,1000,1250,1500,1750]}

tuning = GridSearchCV(estimator =GradientBoostingClassifier(max_depth=4, min_samples_split=2, min_samples_leaf=1, subsample=1,ma
                      param_grid = p_test3, scoring='accuracy',n_jobs=4, cv=5)

tuning.fit(X_train_final,y_train)
tuning.cv_results_, tuning.best_params_,tuning.best_score_
```

```
#MAX_DEPTH
p_test2 = {'max_depth':[2,3,4,5,6,7] }
tuning = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.01,n_estimators=250, min_samples_split=2, min_sample
            param_grid = p_test2, scoring='accuracy',n_jobs=4, cv=5)
tuning.fit(X_train_final,y_train)
tuning.cv_results_, tuning.best_params_,tuning.best_score_
```

```
#MIN SAMPLE SPLIT AND MIN SAMPLES LEAF
p_test4 = {'min_samples_split':[2,4,6,8,10,20,40,60,100], 'min_samples_leaf':[1,3,5,7,9]}

tuning = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.01, n_estimators=250,max_depth=7, subsample=1,max_fe
            param_grid = p_test4, scoring='accuracy',n_jobs=4, cv=5)
tuning.fit(X_train,y_train)
tuning.cv_results_, tuning.best_params_,tuning.best_score_
```

```
#MAX FEATURES
#TUNING MAX FEATURES
p_test5 = {'max_features':[2,3,4,5,6,7]}
tuning = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.01, n_estimators=250,max_depth=7, min_samples_split=
param_grid = p_test5, scoring='accuracy',n_jobs=4, cv=5)
tuning.fit(X_train,y_train)
tuning.cv_results_, tuning.best_params_,tuning.best_score_
```

```
#SUB SAMPLE
p_test6= {'subsample':[0.7,0.75,0.8,0.85,0.9,0.95,1]}

tuning = GridSearchCV(estimator =GradientBoostingClassifier(learning_rate=0.01, n_estimators=250,max_depth=7, min_samples_split=
param_grid = p_test6, scoring='accuracy',n_jobs=4, cv=5)
tuning.fit(X_train,y_train)
tuning.cv_results_, tuning.best_params_,tuning.best_score_
```

Fig 18. Tunning parameters to enhance Gradient Boosting Model

```
#LAST MODEL
from sklearn.model_selection import StratifiedKFold

new=GradientBoostingClassifier(learning_rate=0.01, n_estimators=250,max_depth=7, min_samples_split=40,
                               min_samples_leaf=9,max_features=7 , subsample=1, random_state=10)

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
new.fit(X_train_final,y_train)


#EVALUATE WITH CROSS-VALIDATION
scores = cross_val_score(new, X_train_final, y_train, cv=skf, scoring="accuracy", n_jobs=-1)
scores_test = cross_val_score(new, X_test_final, y_test, scoring='accuracy', cv=skf, n_jobs=-1)

print("cross_validation train accuracy", scores.mean())
print("cross_validation test accuracy", scores_test.mean())


#PREDICT
pred_train=new.predict(X_train_final)
pred=new.predict(X_test_final)

print("Train accuracy::",accuracy_score(y_train,pred_train))
print("Test accuracy::",accuracy_score(y_test,pred))
```

Figure 19. Version of Gradient Boosting model with hyperparameters

```
GB_classifier_SMOTE = GradientBoostingClassifier(learning_rate=0.01, n_estimators=250,max_depth=7, min_samples_split=40,
                                                 min_samples_leaf=9,max_features=7 , subsample=1, random_state=10)

GB_classifier_SMOTE.fit(X_train_SMOTE,y_train_SMOTE)
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)

# Run prediction on test set.

y_predict_train_smote=GB_classifier_SMOTE.predict(X_train_SMOTE)
y_pred_smote = GB_classifier_SMOTE.predict(X_test_final)

y_score_SMOTE_gb = GB_classifier_SMOTE.predict_proba(X_test_final)

#EVALUATE WITH CROSS-VALIDATION
scores_gb_smote = cross_val_score(GB_classifier_SMOTE, X_train_SMOTE, y_train_SMOTE, cv=skf, scoring="accuracy", n_jobs=-1)
scores_test_gb_smote = cross_val_score(GB_classifier_SMOTE, X_test_final, y_test, scoring='accuracy', cv=skf, n_jobs=-1)

print("Train accuracy::",accuracy_score(y_train_SMOTE,y_predict_train_smote))
print("Test accuracy::",accuracy_score(y_test,y_pred_smote))
```

Figure 19. Version of Gradient Boosting model with hyperparameters and SMOTE technique applied in the dataset

In the last part of Gradient Boosting model, it is performed a plot that shows the feature importance using this algorithm, following the same code as Random Forest.

Xtreme Gradient Boosting (XGBoost)

A similar approach to Gradient Boosting is set for XGBoost. Firstly, a baseline model is performed, then, a search of the best parameters is conducted to fine max_depth, min_child_weight, gamma, subsample, colsample_bytree, and reg_alph. The final model is presented in Figure 20.

```
#XGB model with hyperparameters
xgb5 = xgb.XGBClassifier(learning_rate =0.01, n_estimators=5000, max_depth=9,
 min_child_weight=3, gamma=0, subsample=0.8, colsample_bytree=0.7,reg_alpha=0.1,
 objective= 'multi:softmax',num_classes=5 ,nthread=4, scale_pos_weight=1, seed=27)

xgb5.fit(X_train_final,y_train)

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
#Evaluate
preds_xgb5=xgb5.predict(X_test_final)
y_train_pred_xgb5=xgb5.predict(X_train_final)
y_score2 = xgb5.predict_proba(X_test_final)

#EVALUATE WITH CROSS-VALIDATION
scores_xgb5 = cross_val_score(xgb5, X_train_final, y_train, cv=skf, scoring="accuracy", n_jobs=-1)
scores_test_xgb5 = cross_val_score(xgb5, X_test_final, y_test, scoring='accuracy', cv=skf, n_jobs=-1)

print("cross_validation train accuracy", scores_xgb5.mean())
print("cross_validation test accuracy", scores_test_xgb5.mean())


print("Train accuracy::",accuracy_score(y_train,y_train_pred_xgb5))
print("Test accuracy::",accuracy_score(y_test,preds_xgb5))
```

Figure 20. Last version of Xtreme Gradient Boosting model

3.1.4 Evaluation

Random Forest, Gradient Boosting, and XGB are evaluated with cross-validation to prevent overfitting in the models. Once the predicted values are obtained, they are assessed against the real ones. ROC curve is plotted, and Precision, Recall, F1-score, and Accuracy are calculated in a classification report.

```
#Evaluating
cm5=confusion_matrix(y_test,preds_xgb5)
fig,ax=plt.subplots(figsize=(10,5))
ax.matshow(cm5)
plt.title("Confusion matrix",fontsize=20)
plt.ylabel("True values",fontsize=15)
plt.xlabel("False values",fontsize=15)
for (i,j), z in np.ndenumerate(cm5):
    ax.text(j,i,'{:0.1f}'.format(z),ha="center",va="center")
```

Figure 21. Plotting Confusion matrix

```
print(classification_report(y_test,preds_xgb5))

              precision    recall  f1-score   support

         0.0       0.69      0.74      0.71       607
         1.0       0.36      0.29      0.32       268
         2.0       0.48      0.50      0.49       347
         3.0       0.44      0.46      0.45       169
         4.0       0.03      0.03      0.03        39

    accuracy                           0.54      1430
   macro avg       0.40      0.40      0.40      1430
weighted avg       0.53      0.54      0.53      1430
```

Figure 22. Printing classification report

```
#Plot metrics
plot_roc(y_test,y_score2)
plt.show()
```

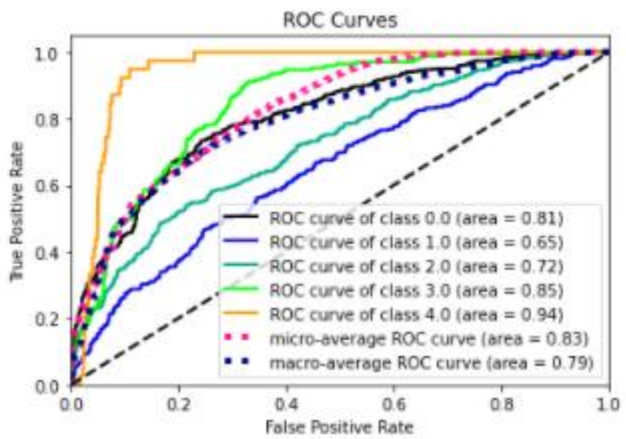

Figure 23. Printing ROC Curve

Deep Learning Models

Deep learning models were implemented on Google Colab due to some advantages as the time required to run the models and memory available in the cloud service. The source code for the deep learning models was based on a GitHub repo[2].

3.1.1 Data Gathering

Images are available on the OAI study website, however, in terms of accessibility and easy management of them, we worked with a dataset that has been already cropped[3] as is illustrated in Figure 24, this dataset corresponds to the baseline of the study. However, the

[2] https://github.com/fontainelam/KneeOsteoarthritis.git
[3] Chen, Pingjun (2018), "Knee Osteoarthritis Severity Grading Dataset", Mendeley Data, V1, doi: 10.17632/56rmx5bjcr.1

split into train, test, and validation of 4,466 knee x-ray images were rearranged according to our train and test dataset from clinical data as is shown in Figure 25.

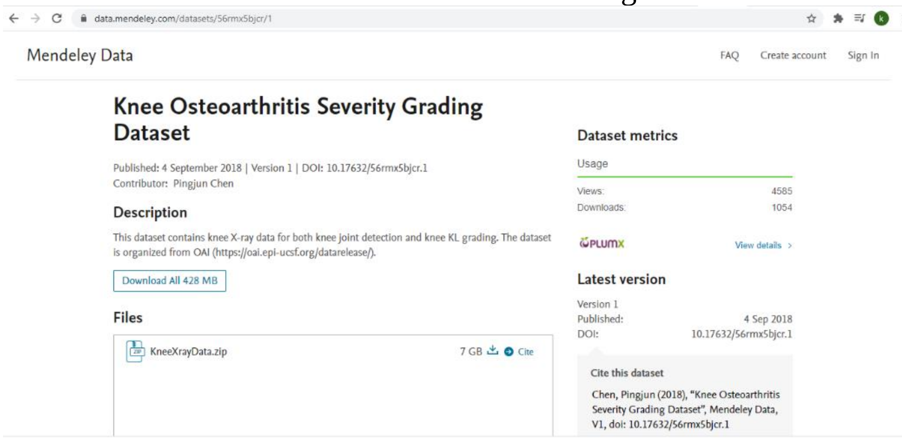

Figure 24. X-ray images Dataset

```
In [208]: for file_name in images_to_consider:
              shutil.move(os.path.join(original,file_name),target)

In [212]: original1 = r"C:\Users\karen\Downloads\archive_kaggle - Copy\KEEPTHEM"+"\\"
          target1 = r"C:\Users\karen\Desktop\train_data\train_images"+"\\"

In [214]: train_to_consider = pd.read_excel("C:/Users/karen/Desktop/train_data/train_id.xlsx")
          train_to_consider
```

Fig 25. Selecting x-ray images according to train and test sets created with clinical data

Train, validation, and test datasets are uploaded in Google Drive to be mounted in Google Colab. Click on the URL and select Gmail account to enter the authorization to proceed.

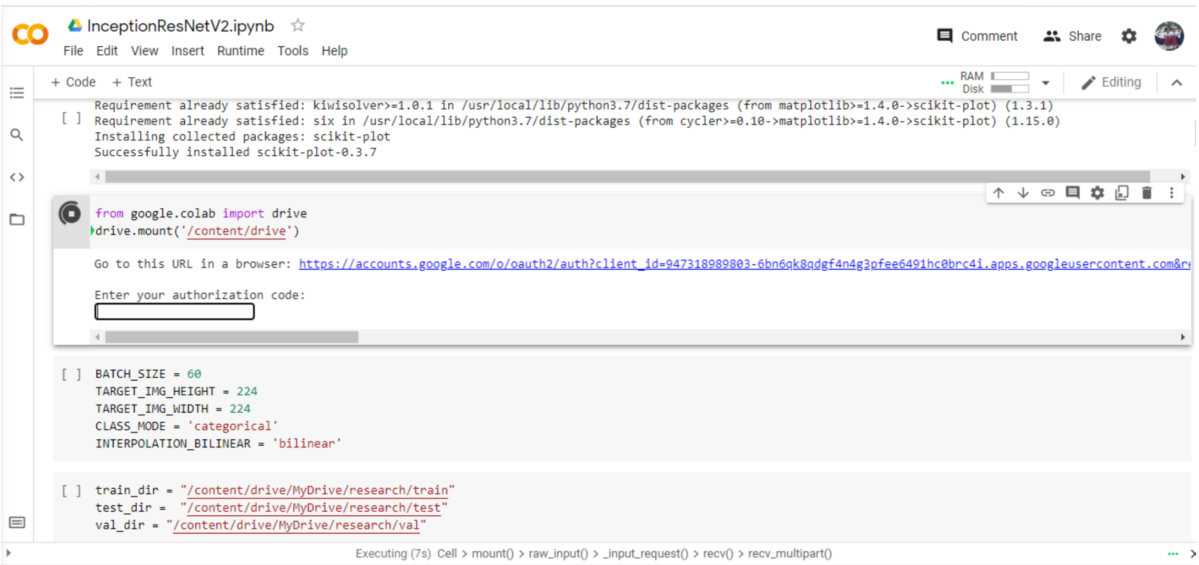

Fig 26. Drive mounted in Jupyter Notebook

```
#load training set
# augment data and shuffle the training dataset
train_generator = get_image_data_from_directory(train_dir, True, True)

#Load validation set
val_generator = get_image_data_from_directory(val_dir)

#load test set
test_generator = get_image_data_from_directory(test_dir)
```

```
Found 3006 images belonging to 5 classes.
Found 330 images belonging to 5 classes.
Found 1430 images belonging to 5 classes.
```

Figure 27. Loading images from Train, Test, and Validation folders

3.1.2 Data Preparation

Figure 28 shows the libraries required to preprocess images by data augmentation. ImageDataGenerator is used from Keras Library.

```
#Data preprocessing
import os
import zipfile
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
```

Fig 28. Libraries required to preprocess x-ray images

Image rotation, Gaussian Blur, horizontal flip, shearing, and zooming are techniques applied to enhance image quality; the script in Figure 29 shows their parameters.

```python
# Creating train, test and validation datasets

def scalar(img): # A customized function to enhance image quality
    img=np.array(img, dtype='uint8')
    img=cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    smooth=cv2.GaussianBlur(img,(3,3),0)
    eql=cv2.equalizeHist(smooth)
    img=cv2.cvtColor(eql, cv2.COLOR_GRAY2RGB)
    img=np.array(img, dtype=('float32'))
    return img


# create a normal, non-augmented data generator
datagen_normal = ImageDataGenerator(preprocessing_function=scalar)

# create an augmented data generator
# vertical flipping, zooming, rotating, shearing, brightness
datagen_augment = ImageDataGenerator(preprocessing_function=scalar,
                                     rotation_range=15,
                                     width_shift_range=0.1,
                                     height_shift_range=0.1,
                                     #brightness_range=[0.3,0.9],
                                     shear_range=0.25,
                                     zoom_range=0.1,
                                     #channel_shift_range = 20,
                                     horizontal_flip = True)
                                     #fill_mode='constant')
```

Figure 29. Generating augmented data

Before implementing the models, the training set is balanced creating synthetic images, The technique will reduce the impact of dealing with an imbalanced dataset.

```python
class_weights_dict = dict(enumerate(class_weight.compute_class_weight(class_weight='balanced', classes=np.asarray(range(5)), y=train_labels_from_files)))
print(class_weights_dict)
```

```
{0: 0.47714285714285715, 1: 1.097080291970803, 2: 0.7952380952380952, 3: 1.5862796833773087, 4: 9.542857142857143}
```

Figure 30. Balancing training dataset

3.1.3 Modelling deep learning methods

The libraries required to implement DenseNet201 and InceptionResNetV2 and its correspondent evaluation are listed in Figure 31.

```python
from tensorflow import keras
from tensorflow.keras import Model, optimizers, preprocessing
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam, Adamax

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.layers.experimental.preprocessing import Rescaling

from tensorflow.keras.applications import DenseNet201

from tensorflow.keras.optimizers import schedules
from tensorflow.keras.optimizers.schedules import PiecewiseConstantDecay

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import class_weight
import cv2
```

Figure 31. Libraries required to implement machine learning models

DenseNet201

All layers in DenseNet201 model are imported from Keras package. A set of parameters such as image size, the rate of dropout, learning rate is declared before running the model. Some layers are stacked at the end of DenseNet201 schema as illustrated in Figure 32.

```
height=224
width=224
img_shape=(height, width, 3)
dropout=.3
lr=.001
img_shape=(height, width, 3)
base_model=tf.keras.applications.DenseNet201( include_top=False, input_shape=img_shape, pooling='max', weights='imagenet')
x=base_model.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(512, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu', kernel_initializer= tf.keras.initializers.GlorotUniform(seed=123))(x)
x=Dropout(rate=dropout, seed=123)(x)
output=Dense(5, activation='softmax',kernel_initializer=tf.keras.initializers.GlorotUniform(seed=123))(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(lr=lr), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 32. Execution of DenseNet201

A callback function is created to manage the performance of the model. Its parameters are set as shown in Figure 33.

```
patience=1      # epochs to wait to lower learning rate is metric does not improve
stop_patience=3  # consecutive epochs to wait without metric improvement before stopping training
threshold=.9    # define level of training accuracy that must be achieved before switching to monitor validation loss
factor=.5       # define value to multiply current learning rate by  to get lower learning rate if metric did not improve
dwell=False     # wxperimental- if metric for current metric did not improve set weights back to those of previous epoch
model_type='DenseNet201'  # model selection name
freeze=True     # if True base model weights are frozen in training
epochs=12       # number of epochs to dun

callbacks=[LRA(model=model,patience=patience,stop_patience=stop_patience, threshold=threshold,
                   factor=factor,dwell=dwell, model_name=model_type, freeze=freeze, end_epoch=epochs - 1 )]
```

Figure 33. Parameters to implement a callback function

The first epochs trained in the previous model are frozen and, 15 layers were added as a strategy to improve the model. The neural network was compiled and executed again.

3.1.4 Evaluation

To track the performance of the neural network, it was plotted accuracy and loss for training and validation data across the epochs, the code is shown in Figure 34. Furthermore, deep learning models followed the same evaluation as machine learning models to be comparable, obtaining ROC curve and metrics such as precision, recall, accuracy, and f1-score.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(total_epochs)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Figure 34. Accuracy and Loss Plot for Training and Validation Sets

InceptionResNetV2

The implementation of this model follows the same process as DenseNet201. The model is called from Keras Package before compiling it.

```
height=224
width=224
img_shape=(height, width, 3)
dropout=.3
lr=.001
img_shape=(height, width, 3)
base_model=tf.keras.applications.InceptionResNetV2( include_top=False, input_shape=img_shape, pooling='max', weights='imagenet')
x=base_model.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(512, kernel_regularizer = regularizers.l2(l = 0.016), activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu', kernel_initializer= tf.keras.initializers.GlorotUniform(seed=123))(x)
x=Dropout(rate=dropout, seed=123)(x)
output=Dense(5, activation='softmax',kernel_initializer=tf.keras.initializers.GlorotUniform(seed=123))(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 35. Execution of InceptionResNetV2

From both models, predicted values in test dataset are exported in a csv. file to complete the last stage in the research project.

4. Fusion Model

This section is implemented in Jupyter Notebook hosted by Anaconda. The first step is to obtain probability scores per class from each machine learning model as is demonstrated in Figure 36. Then, the average between them is calculated and finally, the class with the highest probability score is taken as the KL grade as shown in Figure 38.

```
#From Random Forest
prob_predict = pd.DataFrame(data=y_score_weig,
                            index=X_test.index.copy())

prob_predict.columns = ['RF_0', 'RF_1','RF_2','RF_3','RF4']
prob_predict
```

```
#From Gradient Boosting
prob_predict1 = pd.DataFrame(data=y_score_SMOTE_gb,
                             index=X_test.index.copy())

prob_predict1.columns = ['GB_0', 'GB_1','GB_2','GB_3','GB4']
prob_predict1
```

```
#From Xtreme Gradient Boosting
prob_predict2 = pd.DataFrame(data=y_score2,
                             index=X_test.index.copy())

prob_predict2.columns = ['XGB_0', 'XGB_1','XGB_2','XGB_3','XGB4']
prob_predict2
```

Figure 36. Probability scores per class from ensemble methods

```
#The mean of probaility scores is computed to obtain a single value per class
df_clinicalML['0'] = col_0.mean(axis=1)
df_clinicalML['1'] = col_1.mean(axis=1)
df_clinicalML['2'] = col_2.mean(axis=1)
df_clinicalML['3'] = col_3.mean(axis=1)
df_clinicalML['4'] = col_4.mean(axis=1)
```

Figure 37. Computing means of probability scores per KL grade

```
#Max voting: according to the highest probaility score, it is considered the outcome of that instance
df_clinicalML = df_clinicalML.assign(ML_models=ML_models.values)
df_clinicalML
```

F_3	RF4	GB_0	GB_1	GB_2	GB_3	...	XGB_1	XGB_2	XGB_3	XGB4	0	1	2	3	4	ML_models

Figure 38. Max voting system between machine learning models

A system that performs majority voting between the outcome from machine learning models and predicted values from DenseNet201 and InceptionResNetV2 is implemented, the technique is shown in Figure 39.

```
df_fusion['final'] = df_fusion.apply(lambda row : mode(row['DenseNet201_preds'],
                     row['InceptionResNetV2_preds'], row['ML_models']), axis = 1)
df_fusion
```

	format	ground_truth	DenseNet201_preds	InceptionResNetV2_preds	ML_models	final
ID_SIDE						
9003316L	png	0	1	0	2	1
9003430L	png	0	0	0	1	0
9005321L	png	0	0	0	0	0
9005656L	png	0	0	1	0	0
9006407L	png	0	0	1	0	0

Figure 39. Majority voting between three independent outcomes

To conclude the current research project, metrics to evaluate predicted values against real one is obtained by executing a classification matrix and report.

References

Abedin, J. *et al.* (2019) “Predicting knee osteoarthritis severity: comparative modeling based on patient’s data and plain X-ray images,” *Scientific reports*, 9(1), p. 5761.

Qiu, S. *et al.* (2018) “Fusion of deep learning models of MRI scans, Mini-Mental State Examination, and logical memory test enhances diagnosis of mild cognitive impairment,” *Alzheimer’s & dementia (Amsterdam, Netherlands)*, 10(1), pp. 737–749.

Tiulpin, A. *et al.* (2019) “Multimodal machine learning-based knee osteoarthritis progression prediction from plain radiographs and clinical data,” *Scientific reports*, 9(1), p. 20038.