Configuration Manual

MSc Research Project
Programme Name

Tsai Shih Yang
Student ID: x21101825

School of Computing
National College of Ireland

Supervisor: Dr. Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Tsai Shih Yang
Student ID:	X21101825
Programme:	Data Analytics **Year:** 2022
Module:	Research Project
Lecturer:	10/08/2022
Submission Due Date:	15/08/2022
Project Title:	Recognition and Classification of Knee Osteoporosis and Osteoarthritis Severity using Deep Learning Techniques
Word Count:	1590 **Page Count:** 16

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: Tsai Shih Yang

Date: 10/08/2022

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

Tsai Shih Yang
Student ID: x21101825

1 Introduction

The configuration manual shows the information on hardware and software and all the programming codes we have used in our research project.
The link to our coding in colab:
https://colab.research.google.com/drive/1z8hbJPQbPGl9e1Twkm9G4zHwh955zofv

2 System Configurations

2.1 Hardware

Operation System: Windows 10
Processor: AMD Ryzen 9 4900H with Radeon Graphics 3.30 GHz
Installed RAM: 16.0 GB (15.4 GB usable)

2.2 Software

All the coding parts and the deep learning models were implemented on Google Colab which allowed writing Python language and also allow GPU to be set as a hardware accelerator we have set GPU to increase the compute time but still possibly run out the memory in the higher epochs model test.
The TensorFlow tutorials for google colab link: https://www.tensorflow.org/tutorials

3 Project Implementation

3.1 Kaggle API (kaggle.json)

The Kaggle API can help us to get the dataset we need in this project and the first step is login to your kaggle account and see the profile page then click the account as you can see Figure 1. Create new API token which can download the kaggle.json file we need before we start the coding section on google colab. (Kaggle website: https://www.kaggle.com/)

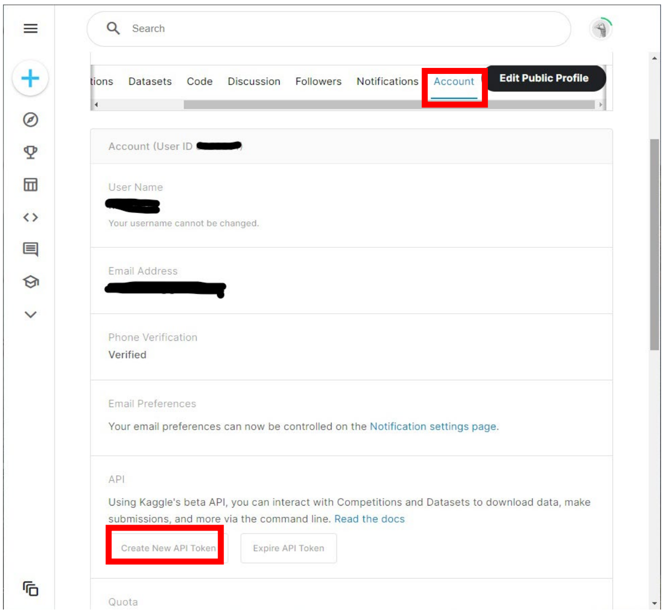


Figure 1: Getting the kaggle.json file from the kaggle website

3.2 Google Colab

In the google colab, we can see the left side shows all the table of contents which can help us to jump into the section you are interested in and show the coding parts. See Figure 2.

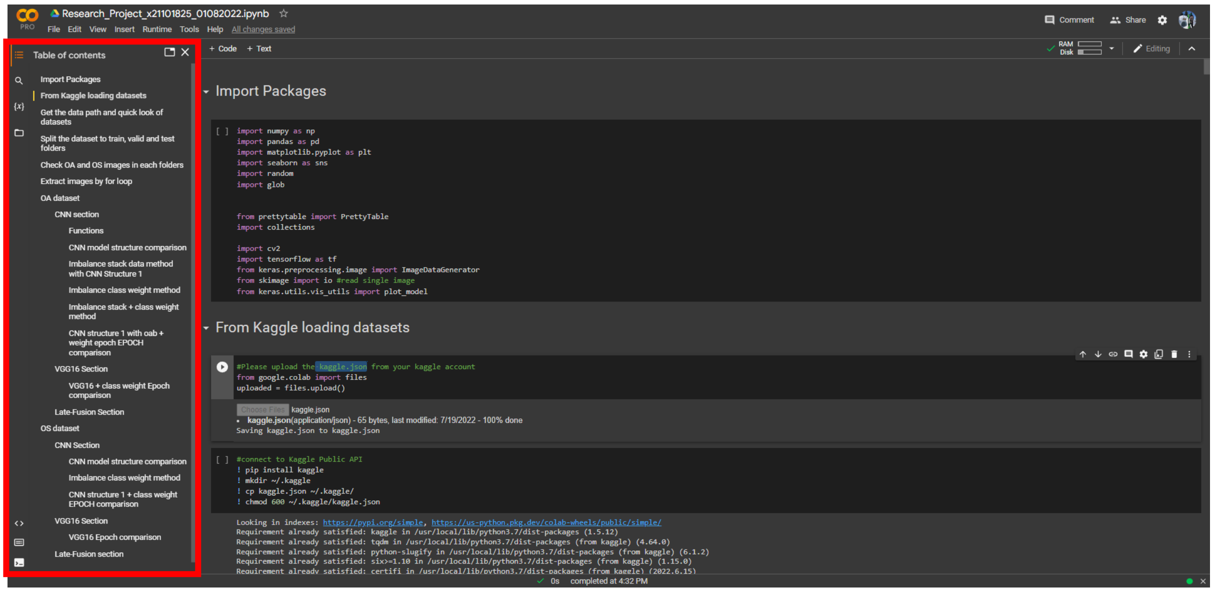


Figure 2: The table of contents in google colab

In the beginning, we can start to install the packages we need and click the run cell can see Figure 3. to install them.

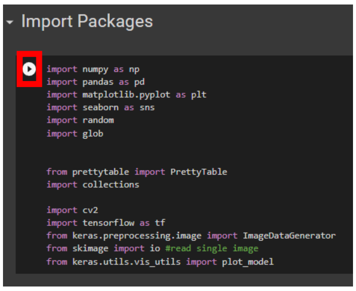


Figure 3: Install package Run cell

3.2.1 From Kaggle loading datasets

We can click the choose files and upload the kaggle.json file we have got from the section 3.1 Kaggle API see Figure 4.

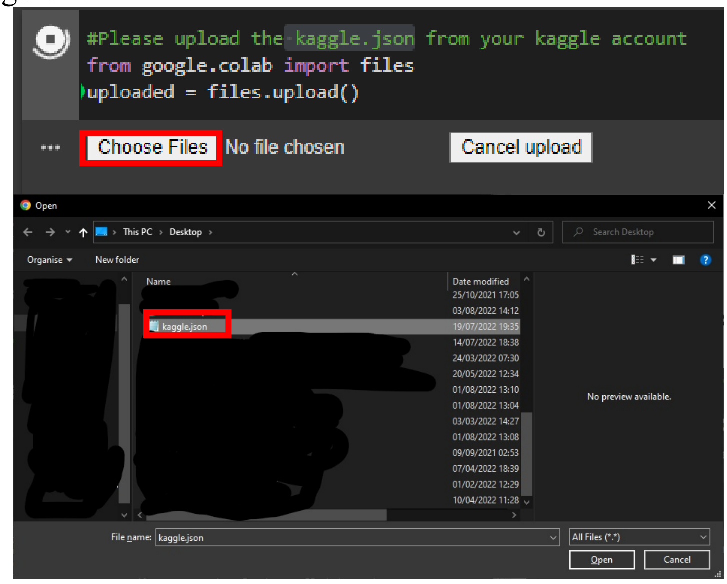


Figure 4: Choose the kaggle.json file

This step is installing kaggle package and connecting with kaggle website by using our kaggle.json file can see Figure 5.

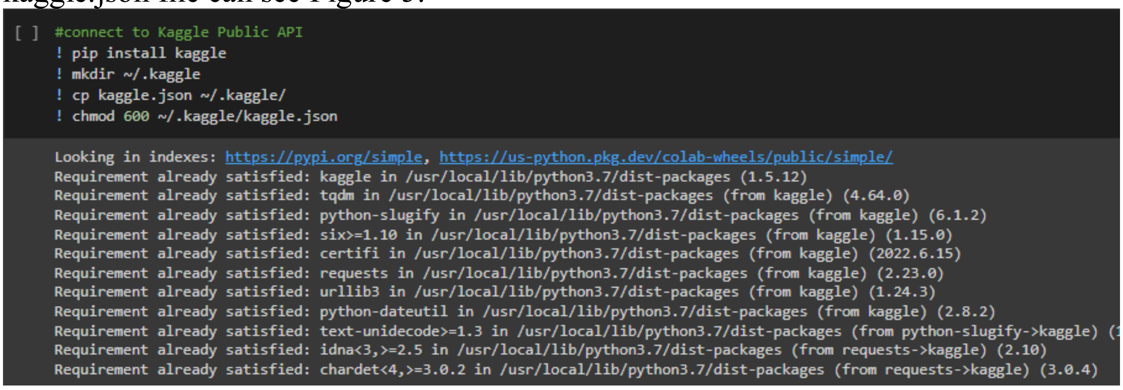
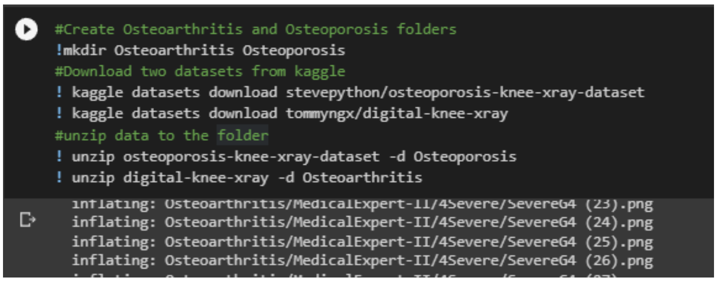


Figure 5: Connecting with kaggle website by using Kaggle API

Figure 6. is showing creating the OA and OS folders and install both dataset and unzip into each folder.

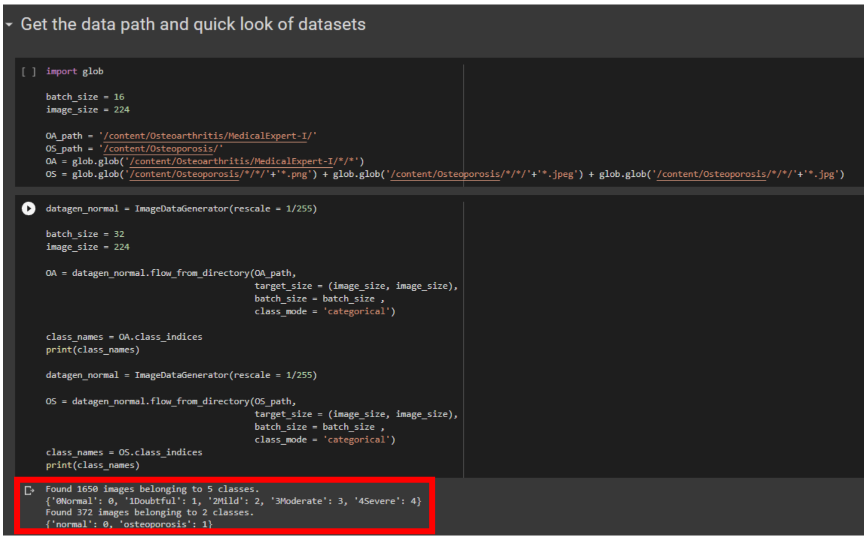


```
#Create Osteoarthritis and Osteoporosis folders
!mkdir Osteoarthritis Osteoporosis
#Download two datasets from kaggle
! kaggle datasets download stevepython/osteoporosis-knee-xray-dataset
! kaggle datasets download tommyngx/digital-knee-xray
#unzip data to the folder
! unzip osteoporosis-knee-xray-dataset -d Osteoporosis
! unzip digital-knee-xray -d Osteoarthritis
```

```
  inflating: Osteoarthritis/MedicalExpert-II/4Severe/SevereG4 (23).png
  inflating: Osteoarthritis/MedicalExpert-II/4Severe/SevereG4 (24).png
  inflating: Osteoarthritis/MedicalExpert-II/4Severe/SevereG4 (25).png
  inflating: Osteoarthritis/MedicalExpert-II/4Severe/SevereG4 (26).png
```

Figure 6: Installing dataset

Figure 7. is using glob to get the images by the path and use ImageDataGenerator to generate images and also we can check the total images and classes we have got from the datasets and we can see OA data (*Digital Knee X-ray*, 2021) has 1650 images with 5 classes and OS data (*Osteoporosis Knee X-ray Dataset | Kaggle*, 2022) has 372 images with 2 classes.



Get the data path and quick look of datasets

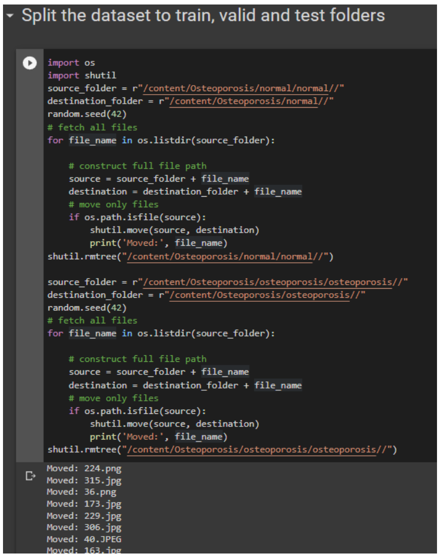
```
import glob

batch_size = 16
image_size = 224

OA_path = '/content/Osteoarthritis/MedicalExpert-I/'
OS_path = '/content/Osteoporosis/'
OA = glob.glob('/content/Osteoarthritis/MedicalExpert-I/*/*')
OS = glob.glob('/content/Osteoporosis/*/*/'+'*.png') + glob.glob('/content/Osteoporosis/*/*/'+'*.jpeg') + glob.glob('/content/Osteoporosis/*/*/'+'*.jpg')
```

```
datagen_normal = ImageDataGenerator(rescale = 1/255)

batch_size = 32
image_size = 224

OA = datagen_normal.flow_from_directory(OA_path,
                                        target_size = (image_size, image_size),
                                        batch_size = batch_size ,
                                        class_mode = 'categorical')

class_names = OA.class_indices
print(class_names)

datagen_normal = ImageDataGenerator(rescale = 1/255)

OS = datagen_normal.flow_from_directory(OS_path,
                                        target_size = (image_size, image_size),
                                        batch_size = batch_size ,
                                        class_mode = 'categorical')
class_names = OS.class_indices
print(class_names)
```

```
Found 1650 images belonging to 5 classes.
{'0Normal': 0, '1Doubtful': 1, '2Mild': 2, '3Moderate': 3, '4Severe': 4}
Found 372 images belonging to 2 classes.
{'normal': 0, 'osteoporosis': 1}
```

Figure 7: Showing each datasets amount of images and classes

We can see the folder path has shown the same name of the classes folder which is not necessary in the OS dataset and we decide to move the image from "/content/Osteoporosis/normal/normal//" to "/content/Osteoporosis/normal//" and the another class so on. See Figure 8.



Split the dataset to train, valid and test folders

```
import os
import shutil
source_folder = r"/content/Osteoporosis/normal/normal//"
destination_folder = r"/content/Osteoporosis/normal//"
random.seed(42)
# fetch all files
for file_name in os.listdir(source_folder):

    # construct full file path
    source = source_folder + file_name
    destination = destination_folder + file_name
    # move only files
    if os.path.isfile(source):
        shutil.move(source, destination)
        print('Moved:', file_name)
shutil.rmtree("/content/Osteoporosis/normal/normal//")

source_folder = r"/content/Osteoporosis/osteoporosis/osteoporosis//"
destination_folder = r"/content/Osteoporosis/osteoporosis//"
random.seed(42)
# fetch all files
for file_name in os.listdir(source_folder):

    # construct full file path
    source = source_folder + file_name
    destination = destination_folder + file_name
    # move only files
    if os.path.isfile(source):
        shutil.move(source, destination)
        print('Moved:', file_name)
shutil.rmtree("/content/Osteoporosis/osteoporosis/osteoporosis//")
```

```
Moved: 224.png
Moved: 315.jpg
Moved: 36.png
Moved: 173.jpg
Moved: 229.jpg
Moved: 306.jpg
Moved: 40.JPEG
Moved: 163.jpg
```

Figure 8: Move image data to the folder we want

Figure. 9 is the dataset split method which can help to split the images from folders to train, valid and test folders by the ratio we provided so we can skip this part which will not influence the coding procedure.

```
#Split the Osteoarthritis and Osteoporosis dataset to train, valid and test 0.7, 0.2, 0.1
!pip install split-folders
import splitfolders

# Split Osteoarthritis dataset
input_folder = '/content/Osteoarthritis/MedicalExpert-I/'
output_folder= '/content/Osteoarthritis/'
splitfolders.ratio(input_folder, output= output_folder, seed=42, ratio = (0.7, 0.2, 0.1))

# Split Osteoporosis dataset
input_folder = '/content/Osteoporosis/'
output_folder= '/content/Osteoporosis/'
splitfolders.ratio(input_folder, output= output_folder, seed=42, ratio = (0.7, 0.2, 0.1))
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
Copying files: 1650 files [00:00, 7310.52 files/s]
Copying files: 372 files [00:00, 1049.48 files/s]
```

Figure 9: Split data folders to train, valid and test folders by ratio (0.7, 0.2 0.1)

Figure. 10 can get each folders amount of images and the total images

Check OA and OS images in each folders

```
#Get the Osteoarthritis train, valid and test folder path and check the amont of data images

oa_train_images = tf.io.gfile.glob('../content/Osteoarthritis/train/*/*')
oa_valid_images = tf.io.gfile.glob('../content/Osteoarthritis/val/*/*')
oa_test_images = tf.io.gfile.glob('../content/Osteoarthritis/test/*/*')

# Check images in each folders
print(f'OA dataset division of 70:20:10')
print(f'Total number of training images = {len(oa_train_images)}')
print(f'Total number of validation images = {len(oa_valid_images)}')
print(f'Total number of test images = {len(oa_test_images)}\n')

# Merge the train and validation and test to see the total images
total_files = oa_train_images
total_files.extend(oa_valid_images)
total_files.extend(oa_test_images)
print(f'Total number of images : train_images + valid_images + test_images  = {len(total_files)}\n')
```

```
OA dataset division of 70:20:10
Total number of training images = 1152
Total number of validation images = 328
Total number of test images = 170

Total number of images : train_images + valid_images + test_images  = 1650
```

```
#Get the Osteoporosis train, valid and test folder path and check the amont of data images
os_train_images = tf.io.gfile.glob('../content/Osteoporosis/train/*/*')
os_valid_images = tf.io.gfile.glob('../content/Osteoporosis/val/*/*')
os_test_images = tf.io.gfile.glob('../content/Osteoporosis/test/*/*')

# Check images in each folders
print(f'OS dataset division of 70:20:10')
print(f'Total number of training images = {len(os_train_images)}')
print(f'Total number of validation images = {len(os_valid_images)}')
print(f'Total number of test images = {len(os_test_images)}\n')

# Check the total images
total_files = os_train_images
total_files.extend(os_valid_images)
total_files.extend(os_test_images)
print(f'Total number of images : train_images + valid_images + test_images  = {len(total_files)}\n')
```

```
OS dataset division of 70:20:10
Total number of training images = 260
Total number of validation images = 74
Total number of test images = 38

Total number of images : train_images + valid_images + test_images  = 372
```

Figure 10: Show the amount of images in each folders by each dataset

3.3 Data preparation

In the Figure 11, we get the images from path and gathering together as list such as the oa_data(Original OA data), oab_data (stacked OA data), oa_label and the os_data and os_label.

```
from glob import glob
normal_knee = glob('/content/Osteoporosis/normal/*')
Osteoporosis = glob('/content/Osteoporosis/osteoporosis/*')
OA_0 = glob('/content/Osteoarthritis/MedicalExpert-I/0Normal/*')
OA_1 = glob('/content/Osteoarthritis/MedicalExpert-I/1Doubtful/*')
OA_2 = glob('/content/Osteoarthritis/MedicalExpert-I/2Mild/*')
OA_3 = glob('/content/Osteoarthritis/MedicalExpert-I/3Moderate/*')
OA_4 = glob('/content/Osteoarthritis/MedicalExpert-I/4Severe/*')

# This is stack data for imbalance of OA classes
OA_2b = glob('/content/Osteoarthritis/MedicalExpert-I/2Mild/*') + glob('/content/Osteoarthritis/MedicalExpert-I/2Mild/*')
OA_3b = glob('/content/Osteoarthritis/MedicalExpert-I/3Moderate/*') + glob('/content/Osteoarthritis/MedicalExpert-I/3Moderate/*')
OA_4b = glob('/content/Osteoarthritis/MedicalExpert-I/4Severe/*') + glob('/content/Osteoarthritis/MedicalExpert-I/4Severe/*')


oa_data = [OA_0,OA_1,OA_2,OA_3,OA_4] # imbalance data
oab_data = [OA_0,OA_1,OA_2b,OA_3b,OA_4b] #stacked data
oa_label = ['OA_0_Normal','OA_1_Doubtful','OA_2_Mild','OA_3_Moderate','OA_4_Severe']

os_data = [normal_knee, Osteoporosis]
os_label = ['normal_knee','Osteoporosis']
```

Figure 11: Gathering images

3.4 Functions

Figure 12. data_separate_label() function we can use this function transfer the image data with labels to the size we need to array and the output will be image dataframe and classes dataframe.

```
#Getting images and labels to array function
def data_separate_label(data,label,size):
  df=[]
  labs=[]
  j = 0
  for i in label:
      if label.index(i)==j:
        for k in range(len(data[j])):
          img = cv2.imread(data[j][k])
          img = cv2.resize(img,(size,size))
          df.append(img)
          labs.append(label.index(i))
      j=j+1
  df=np.array(df)
  labs=np.array(labs)
  return df,labs
```

Figure 12: Transfer the images to an array

Figure 13. is the train_test_valid_split() function can help us split the image dataframes into train, valid and test image dataframes and labels six outputs and the ratio we have decided as 0.7, 0.2 and 0.1.

```
from sklearn.model_selection import train_test_split
#The ratio decide to split
train_ratio = 0.7
validation_ratio = 0.2
test_ratio = 0.10

#Train, Validation, and Test data split function
def train_test_valid_split(data_X, data_Y, validation = True):
  do_validation = validation
  if do_validation==True:
    x_train, x_test, y_train, y_test = train_test_split(data_X, data_Y, test_size=1 - train_ratio, random_state=42)
    x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio), random_state=42)
    return  x_train, y_train, x_val, y_val, x_test, y_test

  else:
    x_train, x_test, y_train, y_test = train_test_split(data_X, data_Y, test_size=test_ratio, random_state=42)
    return  x_train, y_train, x_test, y_test
```

Figure 13: Split Train, valid and test from the image dataframe

Figure 14. is the show_data_table() function which is used PrettyTable to help to show the details of the percentage of our split dataset on each class.

```
#Use pretty table show the data information
def show_data_table(table_name, data_label, train_label,valid_label, test_label):
  # Specify the Column Names while initializing the Table
  myTable = PrettyTable(["Type", "Training Dataset","Validation Dataset","Testing Dataset"],
                         title = f"The composition of {table_name} X-ray dataset")
  for i in range(len(data_label)):
    myTable.add_row([data_label[i],
                      str(collections.Counter(train_label)[i]) + " (" + str(round(collections.Counter(train_label)[i]/len(train_label)*100,1)) + "%)" ,
                      str(collections.Counter(valid_label)[i]) + " (" +  str(round(collections.Counter(valid_label)[i]/len(valid_label)*100,1)) + "%)",
                      str(collections.Counter(test_label)[i]) + " (" +  str(round(collections.Counter(test_label)[i]/len(test_label)*100,1)) + "%)"])

  myTable.add_row(["Total images",
                    str(len(train_label)) + " (" +  str(round(len(train_label)/len(train_label)*100,1)) + "%)",
                    str(len(valid_label)) + " (" +  str(round(len(valid_label)/len(valid_label)*100,1)) + "%)",
                    str(len(test_label)) + " (" +  str(round(len(test_label)/len(test_label)*100,1)) + "%)"])
  print(myTable)
```

Figure 14: Show the composition of the dataset

Figure 15. is the EDA_data() function which can get the countplot from the train, valid and test dataframe and the images amounts in each class.

```python
#Get the countplot of data with each classes
def EDA_data(train_label,valid_label,test_label,valid=True):
  plt.figure(figsize = (17,8));
  if valid:
    lis = ['Train','Valid','Test']
    for i,j in enumerate([train_label,valid_label, test_label]):
      plt.subplot(1,3, i+1);
      ax = sns.countplot(x = j);
      for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.2, p.get_height()+0.5),size = 15)
      plt.xlabel(lis[i])
  else:
    lis = ['Train', 'Test']
    for i,j in enumerate([train_label, test_label]):
      plt.subplot(1,3, i+1);
      ax = sns.countplot(x = j);
      plt.xlabel(lis[i])
      for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.2, p.get_height()+0.5),size = 15)
      plt.xlabel(lis[i])
```

Figure 15: Show the countplot from image dataframe

Figure 16. is the show_images() function which can show 5 images from the target image dataframe

```python
#This function can show 5 images from the image array
def show_images(img_arr):
    fig, axes = plt.subplots(1, 5, figsize=(10,10))
    axes = axes.flatten()
    for img, ax in zip( img_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

Figure 16: Show countplot from image dataframe

3.5 Knee Osteoarthritis severity dataset (OA dataset)

Using these functions to get the OA image dataframe and labels with size 224 x 224 from the function and split to train, valid and test dataframe and shwo the data table and count plot of data can see Figure 17. The table shows train, valid and test dataframe images with percentages and countplot to know the distribution of data.

OA dataset

```python
[15] oa,oa_l=data_separate_label(oa_data,oa_label,224)
     oab,oab_l=data_separate_label(oab_data,oa_label,224)

[16] train_oa,train_oa_l,valid_oa,valid_oa_l,test_oa,test_oa_l=train_test_valid_split(oa,oa_l, True)

show_data_table("OA severity",oa_label,train_oa_l,valid_oa_l,test_oa_l)
EDA_data(train_oa_l,valid_oa_l,test_oa_l,True)
```

The composition of OA severity X-ray dataset

Type	Training Dataset	Validation Dataset	Testing Dataset
OA_0_Normal	361 (31.3%)	102 (30.9%)	51 (30.7%)
OA_1_Doubtful	342 (29.6%)	90 (27.3%)	45 (27.1%)
OA_2_Mild	157 (13.6%)	46 (13.9%)	29 (17.5%)
OA_3_Moderate	148 (12.8%)	54 (16.4%)	19 (11.4%)
OA_4_Severe	146 (12.7%)	38 (11.5%)	22 (13.3%)
Total images	1154 (100.0%)	330 (100.0%)	166 (100.0%)

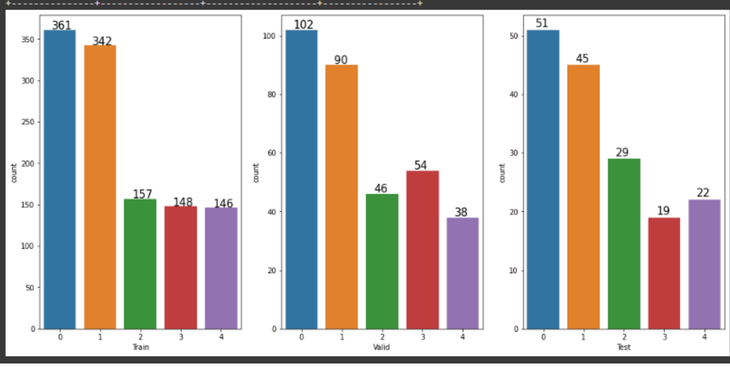


Figure 17: OA data table and distribution

Figure 18. is showing 5 sample images from OA train data by using the function show_images().

Figure 18: OA sample images

3.5.1 CNN structures

The CNN_Structure_1() function is based on (Bany Muhammad *et al.*, 2019) their Network architecture of the base model-2 to build up and we can use this for the CNN model can see Figure 19.

CNN section

Functions

```
from keras import models
from keras import layers
from keras.layers import  Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#(Bany Muhammad et al., 2019 Network architecture of the base model-2.)
def CNN_Structure_1(data_label,activation="relu"):
  n_class = len(pd.Categorical(data_label).categories)
  model = models.Sequential()
  model.add(layers.Conv2D(32, (3, 3), activation=activation, input_shape=(224, 224, 3)))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(96, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(128, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(256, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(Dropout(0.1))
  #Convert the matrix to a fully connected layer
  model.add(layers.Flatten())
  model.add(Dropout(0.1))
  #Final dense convert to classes
  model.add(layers.Dense(n_class, activation='softmax'))
  #Model compile
  model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
  return model
```

Figure 19: CNN structure 1

Figure 20. is the CNN_Structure_2() function is based on (Nafiiyah and Setyati, 2021) their 2021 CNN 35 Layers Architecture to build up.

```
#(Nafiiyah and Setyati, 2021 CNN 35 Layers Architecture )
def CNN_Structure_2(data_label,activation="relu"):
  n_class = len(pd.Categorical(data_label).categories)
  model = models.Sequential()
  model.add(layers.Conv2D(32, (3, 3), activation=activation, input_shape=(224, 224, 3)))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(32, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(128, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(128, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((1, 1)))
  model.add(layers.Conv2D(256, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((1, 1)))
  #Convert the matrix to a fully connected layer
  model.add(layers.Flatten())
  #Final dense convert to classes
  model.add(layers.Dense(n_class, activation='softmax'))
  #Model compile
  model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
  return model
```

Figure 20: CNN structure 2

Figure 21. is the CNN_Structure_3() function is we have built up by our own.

```python
def CNN_Structure_3(data_label,activation="relu"):
  n_class = len(pd.Categorical(data_label).categories)
  model = models.Sequential()
  model.add(layers.Conv2D(32, (3, 3), activation=activation, input_shape=(224, 224, 3)))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation=activation))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(128, (3, 3), activation=activation))
  #Convert the matrix to a fully connected layer
  model.add(layers.Flatten())
  #Add dense layers
  model.add(layers.Dense(256, activation="relu"))
  model.add(layers.Dense(512, activation="relu"))
  #Final dense convert to classes
  model.add(layers.Dense(n_class, activation='softmax'))

  model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
  return model
```

Figure 21: CNN structure 3

Figure 22. We can use these two function to run our CNN model with class weight and without class weight after we have built up the CNN structure to get the model.

```python
#These function can test different epochs of CNN model
def CNN_Model(model,train_data, train_label,valid_data,valid_label, epochs):
  random.seed(42)
  history = model.fit(train_data, train_label, epochs=epochs, validation_data=(valid_data, valid_label),verbose=0)
  return history

#This function can add the class_weight parameters and adjust epochs
def CNN_Model_w(model,train_data, train_label,valid_data,valid_label, epochs, class_weight):
  random.seed(42)
  history = model.fit(train_data, train_label, epochs=epochs,class_weight = class_weight, validation_data=(valid_data, valid_label),verbose=0)
  return history
```

Figure 22: CNN model with class weight and without

Figure 23. We can use the history of training model to check the accuracy with the epochs and show the plot of their relation by this acc_plot() function.

```python
#Show the epochs with accuracy relation plot
def acc_plot(history):
  fig, ax = plt.subplots(1, 2, figsize=(10, 3))
  ax = ax.ravel()

  for i, met in enumerate(['accuracy', 'loss']):
      ax[i].plot(history.history[met])
      ax[i].plot(history.history['val_' + met])
      ax[i].set_title('Model {}'.format(met))
      ax[i].set_xlabel('epochs')
      ax[i].set_ylabel(met)
      ax[i].legend(['train', 'val'])
```

Figure 23: Show accuracy plot function of model

Figure 24. The show_matrix() function can use our train model with our test data to get the confusion matrix by using heatmap and the matrix_info() function can show the information of matrix which shows precision, recall, f1-score, and accuracy.

```python
#This show_matrix funciton can show confusion matrix by heatmap
def show_matrix(model,test_data,test_data_label,label_names):
    predict=model.predict(test_data)
    predict_labels=np.argmax(predict,axis=1)
    #Generate the confusion matrix
    cf_matrix = confusion_matrix(test_data_label,predict_labels)

    plt.figure(figsize = (8,6))
    ax = sns.heatmap(cf_matrix, annot=True, annot_kws={'size': 20}, cmap='Blues',fmt='g')

    ax.set_title('Confusion Matrix',size= 20);
    ax.set_xlabel('\nPredicted')
    ax.set_ylabel('Actual');
    ax.xaxis.set_ticklabels(label_names,size=10)
    ax.yaxis.set_ticklabels(label_names,size=10, rotation=45)
    ## Display the visualization of the Confusion Matrix.
    plt.show()
    print("\n")

#This funciton can show print the information of matrix
from sklearn.metrics import classification_report
def matrix_info(model,test_data,test_label):
  test_pred=np.argmax(model.predict(test_data),axis=1)
  print(f"\n{classification_report(test_label,test_pred)}")
```

Figure 24: The function of confusion matrix with matrix information

Figure 25.We have run three CNN structure and input the parameters for the function needs and the result can see Figure 26. The CNN structure 1 gets 49% accuracy

CNN model structure comparison

```
#CNN structure 1
oa_cnn_model1 = CNN_Structure_1(train_oa_l,"relu")
oa_history1 = CNN_Model(oa_cnn_model1,train_oa,train_oa_l,valid_oa,valid_oa_l,20)
acc_plot(oa_history1)
show_matrix(oa_cnn_model1,test_oa,test_oa_l,oa_label)
matrix_info(oa_cnn_model1,test_oa,test_oa_l)

#CNN structure 2
oa_cnn_model2 = CNN_Structure_2(train_oa_l,"relu")
oa_history2 = CNN_Model(oa_cnn_model2,train_oa,train_oa_l,valid_oa,valid_oa_l,20)
acc_plot(oa_history2)
show_matrix(oa_cnn_model2,test_oa,test_oa_l,oa_label)
matrix_info(oa_cnn_model2,test_oa,test_oa_l)

#CNN structure 3
oa_cnn_model3 = CNN_Structure_3(train_oa_l,"relu")
oa_history3 = CNN_Model(oa_cnn_model3,train_oa,train_oa_l,valid_oa,valid_oa_l,20)
acc_plot(oa_history3)
show_matrix(oa_cnn_model3,test_oa,test_oa_l,oa_label)
matrix_info(oa_cnn_model3,test_oa,test_oa_l)
```

Figure 25. Run these 3 CNN structures with OA data

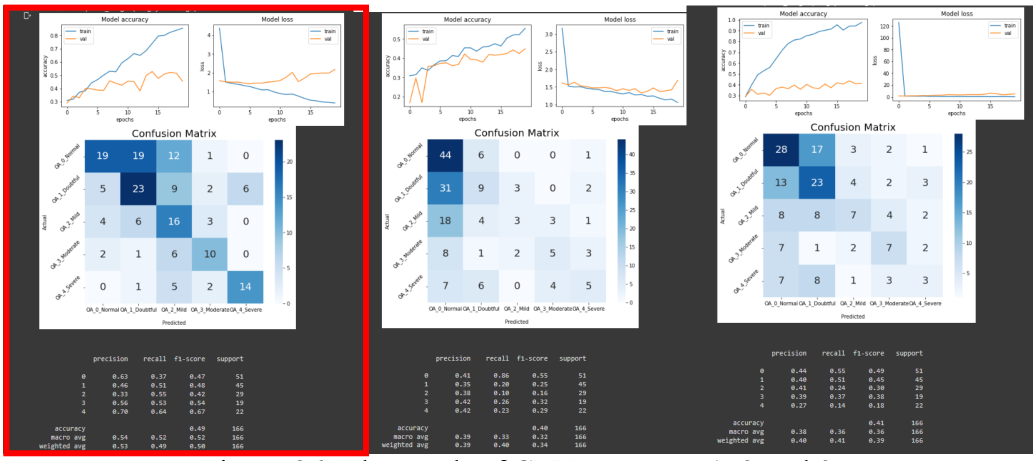


Figure 26. The result of CNN structure 1, 2 and 3

Figure 27.We check the stacked OA data of data information and distribution before implement to CNN structure 1.

Imbalance stack data method with CNN Structure 1

```
train_oab,train_oab_l,valid_oab,valid_oab_l,test_oab,test_oab_l =train_test_valid_split(oab,oab_l, True)
```

```
show_data_table("OA severity",oa_label,train_oab_l,valid_oab_l,test_oab_l)
EDA_data(train_oab_l,valid_oab_l,test_oab_l,True)
```

The composition of OA severity X-ray dataset

Type	Training Dataset	Validation Dataset	Testing Dataset
OA_0_Normal	348 (21.5%)	109 (23.6%)	57 (24.7%)
OA_1_Doubtful	338 (20.9%)	95 (20.6%)	44 (19.0%)
OA_2_Mild	328 (20.3%)	91 (19.7%)	45 (19.5%)
OA_3_Moderate	320 (19.8%)	79 (17.1%)	43 (18.6%)
OA_4_Severe	282 (17.5%)	88 (19.0%)	42 (18.2%)
Total images	1616 (100.0%)	462 (100.0%)	231 (100.0%)

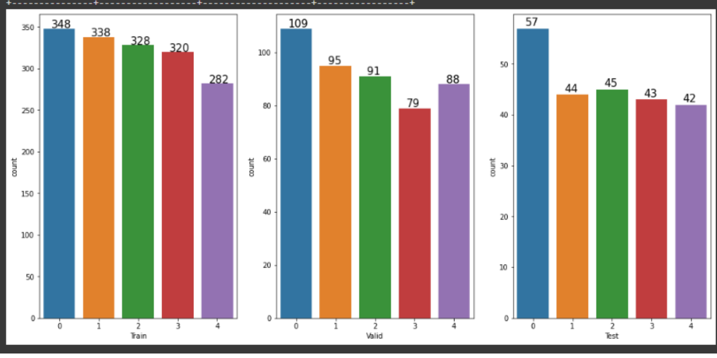


Figure 27. The OA data information table with distribution

Figure 28.We use the CNN structure 1 with stacked OA data

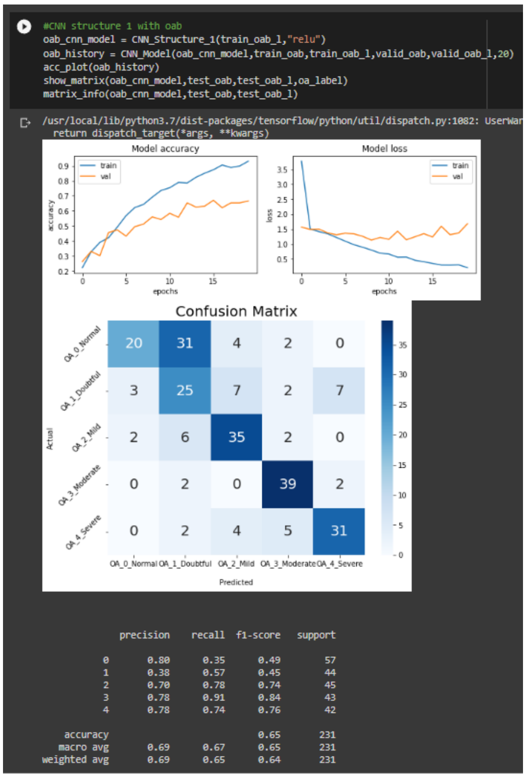


Figure 28. The stacked OA data by using CNN structure 1

Figure 29. is the generate_class_weights() function which can help us to compute the class weight and we can use this for our train model.

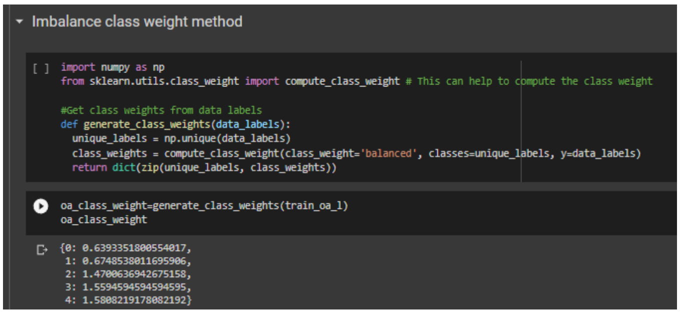


Figure 29. The class weight function

Figure 30. is the result of stacked OA data with the class weight and use the CNN structure 1 we have choose and the result of accuracy shows 74% also we have compared the Epochs 50 and 100 the detail can see on our colab code link we have provided which can see in the section 1 introduction.

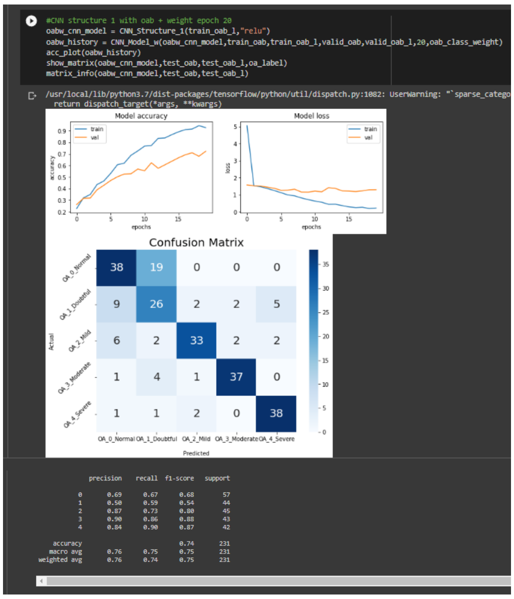


Figure 30. The stacked OA data with class weight by using CNN structure 1

3.5.2 VGG16 Function

Figure 31. is showing our VGG16 base model function which can run the data we will provide and also we have check the result of accuracy and confustion matrix.

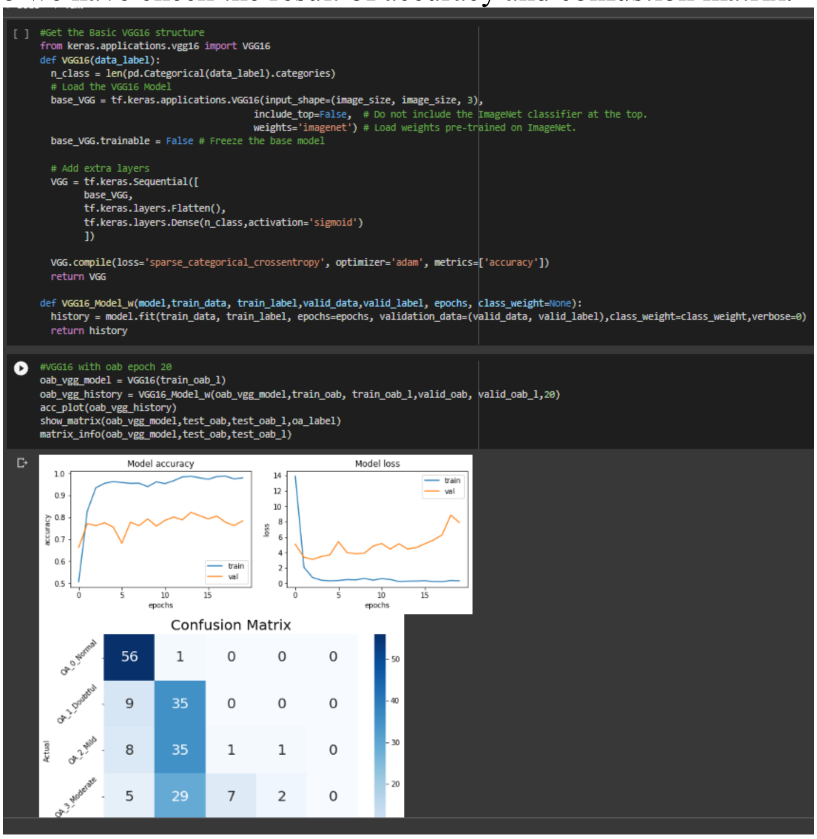


Figure 31. VGG16 structure and the model result

3.5.3 Late-Fusion model

Figure 32. is our Late-Fusion model which combine the CNN structure 1 with VGG16 structure to build up.

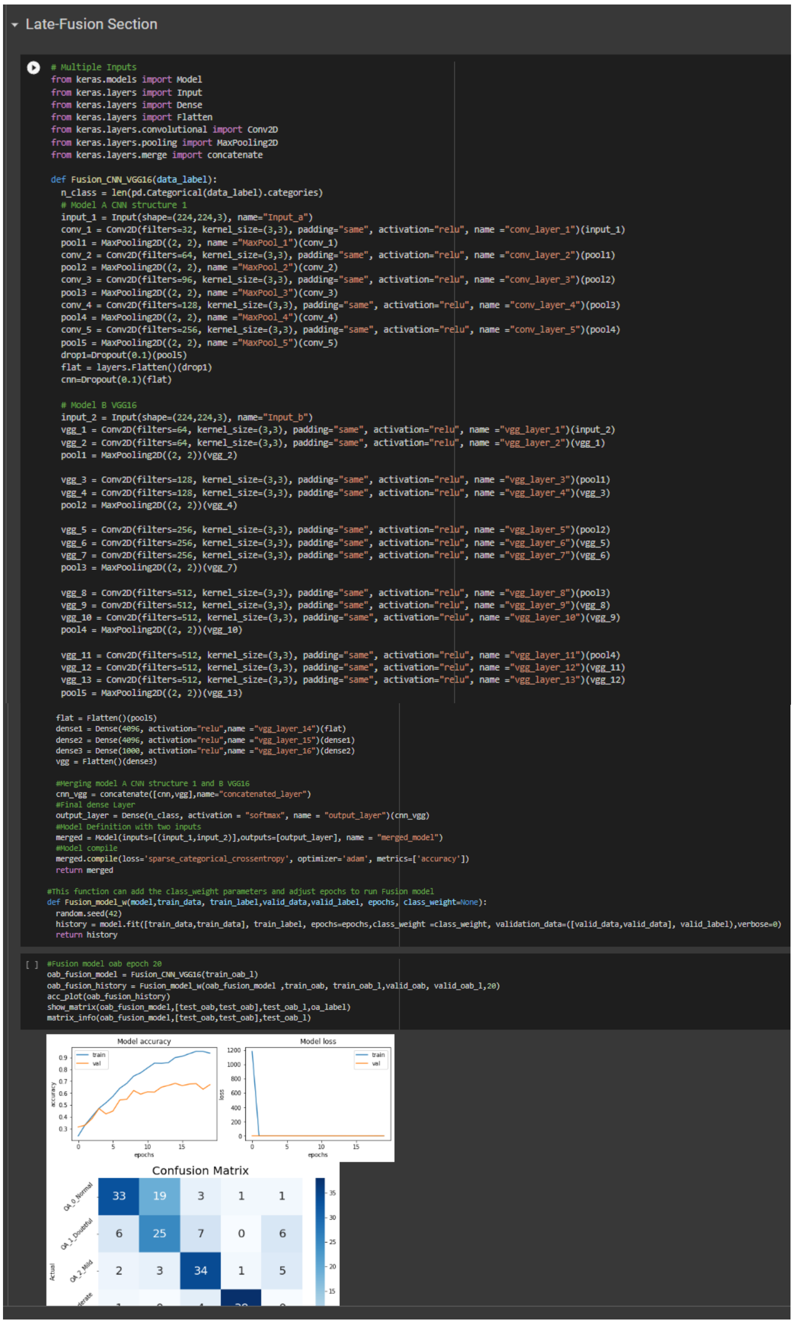


Figure 32. Late Fusion model with the result of stacked data

3.6 Knee Osteoporosis dataset (OS dataset)

Figure 33. Using the functions we have built to check the OS data information and some sample images.

Figure 33. OS dataset information table, countplots and sample images

3.6.1 CNN

Figure 34. can see the our OS data with the CNN structure 1 model and the result of performance.

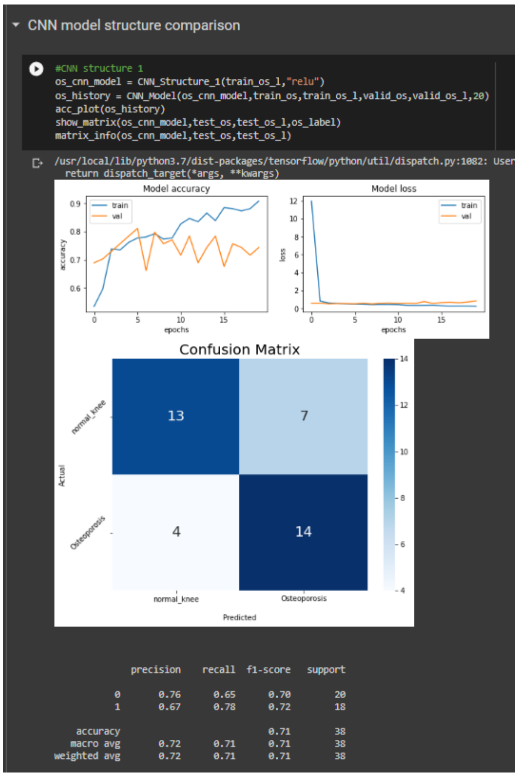


Figure 34. OS dataset with CNN structure 1

3.6.2 VGG16

Figure 35. can see the our OS data with the VGG16 model and the result of performance.

Figure 35. OS dataset with VGG16

3.6.3 Late Fusion

Figure 36. can see the our OS data with the VGG16 model and the result of performance.

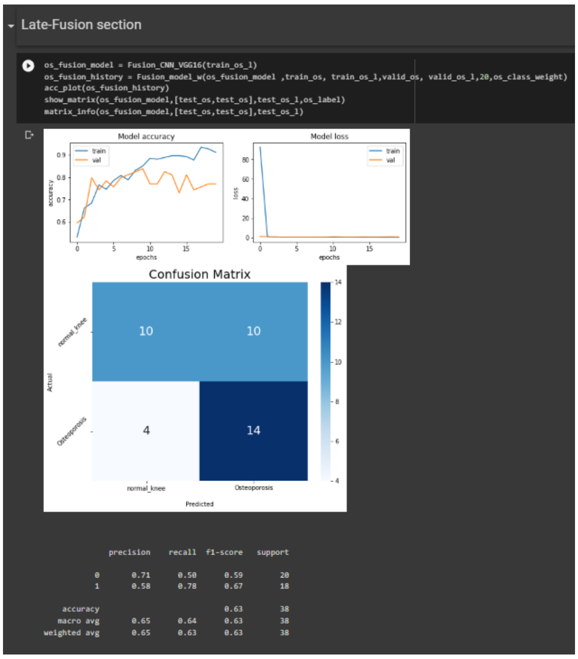


Figure 36. OS dataset with VGG16

References

Bany Muhammad, M. *et al.* (2019) ‘Deep ensemble network for quantification and severity assessment of knee osteoarthritis’, *Proceedings - 18th IEEE International Conference on Machine Learning and Applications, ICMLA 2019*, pp. 951–957. doi: 10.1109/ICMLA.2019.00163.

Digital Knee X-ray (2021). Available at: https://www.kaggle.com/datasets/tommyngx/digital-knee-xray (Accessed: 15 March 2022).

Nafiiyah, N. and Setyati, E. (2021) ‘Lung X-Ray Image Enhancement to Identify Pneumonia with CNN’, *3rd 2021 East Indonesia Conference on Computer and Information Technology, EIConCIT 2021*, pp. 421–426. doi: 10.1109/EIConCIT50028.2021.9431856.

Osteoporosis Knee X-ray Dataset | Kaggle (2022). Available at: https://www.kaggle.com/stevepython/osteoporosis-knee-xray-dataset (Accessed: 15 March 2022).