TEAM ID	**PNT2022TMID01315**
PROJECT NAME	**Classification of arrhythmia by using Deep learning with 2-D ECG Spectral Image Representation**
DATE	**23 Nov 2022**

CODING AND SOLUTIONING

Dataset Collection

The dataset contains six classes:

1. Left BundleBranch Block
2. Normal
3. Premature AtrialContraction
4. Premature Ventricular Contractions
5. Right BundleBranch Block
6. Vtricular Fibrillationen

Image Preprocessing:

Image Pre-processing includesthe following main tasks

a.Import ImageDataGenerator Library:

Image data augmentation is a techniquethat can be used to artificially expand the

size of a training dataset by creating modified versions of images in the dataset. The Keras deep learning neuralnetwork library providesthe capability to fit modelsusing image data augmentation via the ImageDataGenerator class.

```
In [5]:  1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

b.Configure ImageDataGenerator Class:

There are five main types of data augmentation techniques for image data; specifically:

1. Image shifts via the width_shift_range and height_shift_range arguments.
2. Image flips via the horizontal_flip and vertical_flip arguments.

3. Image rotates via the rotation_range argument
4. Image brightness via the brightness_range argument.
5. Image zooms via the zoom_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

```
In [6]:
train_datagen = ImageDataGenerator(rescale = 1./255,shear_range = 0.2,zoom_range = 0.2,horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

c.Applying ImageDataGenerator functionality to the trainset and test set:

We will apply ImageDataGenerator functionality to Trainset and Testset by using the following code

This function will return batches of images from the subdirectories Left Bundle Branch Block, Normal, Premature Atrial Contraction, Premature Ventricular Contractions, Right Bundle Branch Block and Ventricular Fibrillation, together with labels 0 to 5{'Left Bundle Branch Block': 0, 'Normal': 1, 'Premature Atrial Contraction': 2, 'Premature Ventricular Contractions': 3, 'Right Bundle Branch Block': 4, 'Ventricular Fibrillation': 5}

```
In [7]:
x_train = train_datagen.flow_from_directory("/content/data/train",target_size = (64,64),batch_size = 32,\
                                            class_mode = "categorical")
x_test = test_datagen.flow_from_directory("/content/data/test",target_size = (64,64),batch_size = 32,\
                                          class_mode = "categorical")

Found 15341 images belonging to 6 classes.
Found 6825 images belonging to 6 classes.
```

We can see that for training there are 15341 images belonging to 6 classes and for testing there are 6825 images belonging to 6 classes.

Model Building

We are ready with the augmented and pre-processed image data,we will begin our build our model by following the below steps:

a.Import the model building Libraries:

```
In [4]:
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
```

b. Initializing the model:

Keras has 2 ways to define a neural network:

1. Sequential

2. Function API

The Sequential class is used to define linear initializations of network layers which then,collectively, constitute a model. In our examplebelow, we will use the Sequential constructor to create a model, which will then have layers added to it using the add ()method.
Now, will initialize our model.

1.Adding CNN Layers:

We are adding a convolution layer with an activation functionas “relu” and with a small filtersize (3,3) and a number of filters as (32) followed by a max-pooling layer.

The Max pool layer is used to downsample the input.
The flatten layer flattens the input.

```
In [9]:  #MODEL BUILDING

In [10]: model = Sequential()

In [11]: model.add(Convolution2D(32,(3,3),input_shape = (64,64,3),activation = "relu"))

In [12]: model.add(MaxPooling2D(pool_size = (2,2)))

In [13]: model.add(Convolution2D(32,(3,3),activation='relu'))

In [14]: model.add(MaxPooling2D(pool_size=(2,2)))

In [15]: model.add(Flatten()) # ANN Input...
```

2.Adding Hidden Layer

Dense layer is deeply connected neuralnetwork layer. It is most common and frequently used layerDense layer is deeply connected neuralnetwork layer.

```
In [16]: #Adding Dense Layers

In [17]: model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [18]: model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [19]: model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [20]: model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [21]: model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))
```

3. Adding Output Layer

```
In [22]: model.add(Dense(units = 6,kernel_initializer = "random_uniform",activation = "softmax"))
```

Understanding the model is very important phase to properly use it for trainingand prediction purposes.Keras provides a simple method, summary to get the full information about the model and its layers.

In [23]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 128)               16512

 dense_5 (Dense)             (None, 6)                 774

=================================================================
Total params: 879,910
Trainable params: 879,910
Non-trainable params: 0
_________________________________________________________________
```

4. Configure the Learning Process

1. The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find error or deviation in the learning process. Keras requires loss function during the model compilation process.
2. Optimization is an importantprocess that optimizesthe input weightsby comparing the prediction and the loss function. Here we are using adam optimizer
3. Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in the training process.

5.Train the Model

We will train our model with our image dataset. fit_generator functions used to train a deep learning neural network

```
In [25]: model.fit_generator(generator=x_train,steps_per_epoch = len(x_train), epochs=9, validation_data=x_test,\
                             validation_steps = len(x_test))
```

```
Epoch 1/9
480/480 [==============================] - 99s 203ms/step - loss: 1.4415 - accuracy: 0.4788 - val_loss: 1.6093 - val_accurac
y: 0.3193
Epoch 2/9
480/480 [==============================] - 96s 201ms/step - loss: 0.9465 - accuracy: 0.6495 - val_loss: 1.3444 - val_accurac
y: 0.5121
Epoch 3/9
480/480 [==============================] - 97s 201ms/step - loss: 0.5540 - accuracy: 0.8018 - val_loss: 0.7785 - val_accurac
y: 0.7698
Epoch 4/9
480/480 [==============================] - 99s 205ms/step - loss: 0.2770 - accuracy: 0.9069 - val_loss: 0.6690 - val_accurac
y: 0.8296
Epoch 5/9
480/480 [==============================] - 97s 201ms/step - loss: 0.2037 - accuracy: 0.9388 - val_loss: 0.6057 - val_accurac
y: 0.8416
Epoch 6/9
 91/480 [====>.........................] - ETA: 1:09 - loss: 0.1595 - accuracy: 0.9499
```

6. Saving the model:

1. The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF).It contains multidimensional arrays of scientific data.

```
In [26]: #Saving Model.
         model.save('ECG.h5')
```

7.Testing the model:

Load necessary libraries and load the saved model using load_model

Taking an image as inputand checking the results

Note: The target size should for the image that is should be the same as the target size that you have used for training.

In [26]:
```
#Saving Model.
model.save('ECG.h5')
```

In [28]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

In [29]:
```
model=load_model('ECG.h5')
```

In [30]:
```
img=image.load_img("/content/Unknown_image.png",target_size=(64,64))
```

In [31]:
```
x=image.img_to_array(img)
```

In [32]:
```
import numpy as np
```

In [33]:
```
x=np.expand_dims(x,axis=0)
```

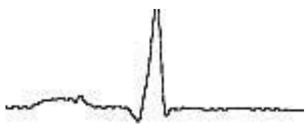
In [34]:
```
pred = model.predict(x)
y_pred=np.argmax(pred)
y_pred
```

Out[34]:
```
1
```

In [35]:
```
index=['left Bundle Branch block',
       'Normal',
       'Premature Atrial Contraction',
       'Premature Ventricular Contraction',
       'Right Bundle Branch Block',
       'Ventricular Fibrillation']
result = str(index[y_pred])
result
```

Out[35]:
```
'Normal'
```

The unknown image uploaded is:



Here the output for the uploaded result is normal

Application Building:

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.

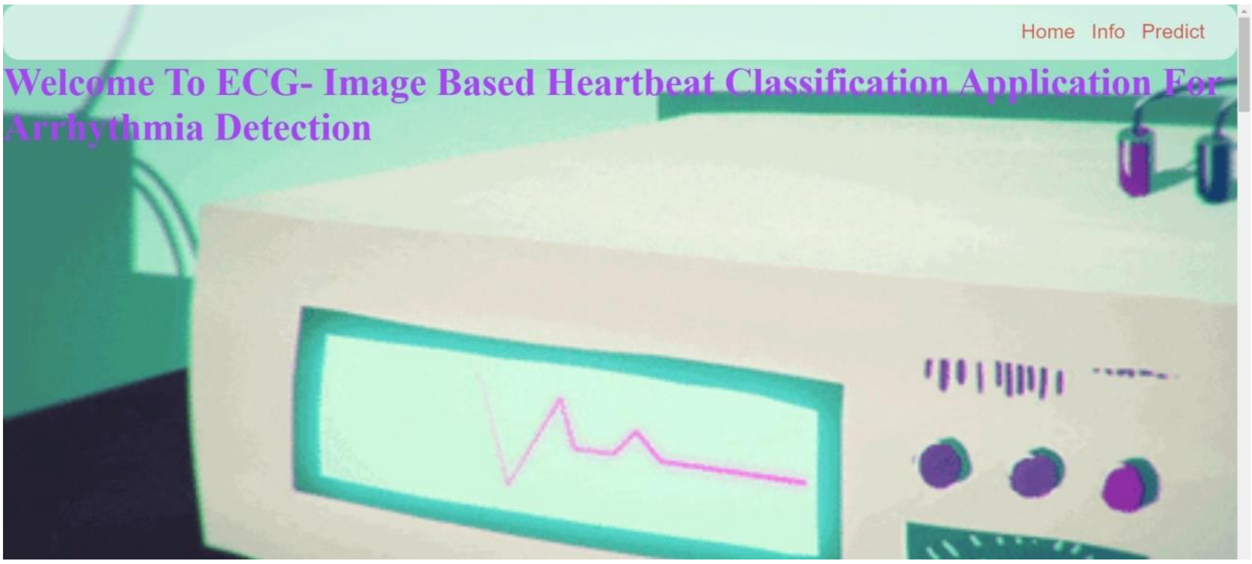
This sectionhas the following tasks

1. Building HTML Pages:

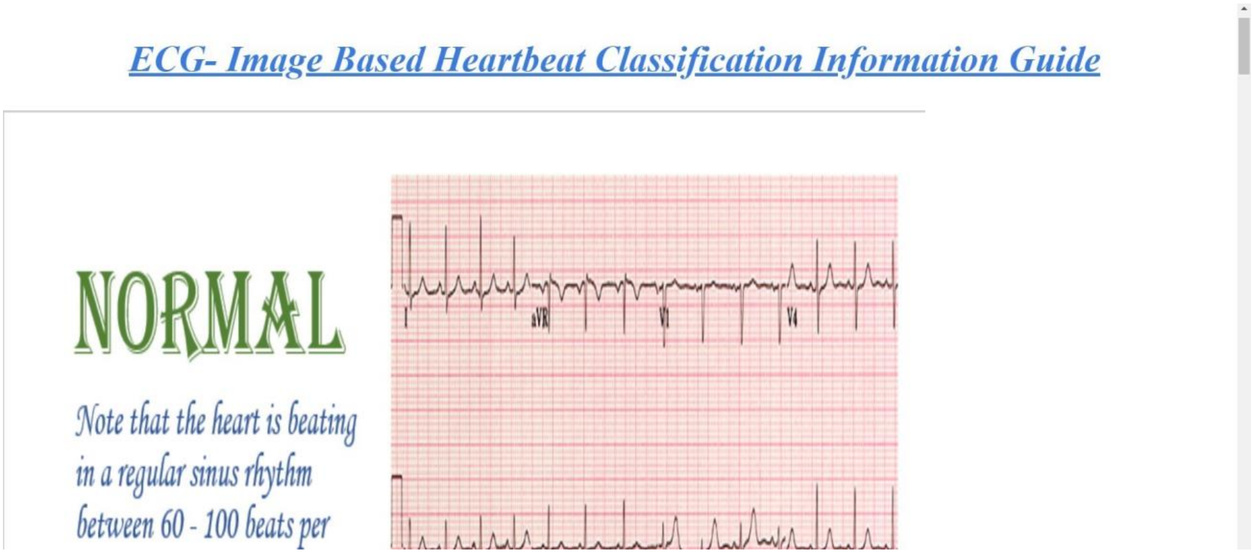
2. Weuse HTML to create the front end part of the web page.

3.Here, we created 4 html pages- home.html, predict_base.html, predict.html, information.html.

4.home.html displays the home page
home.html displays the home page.



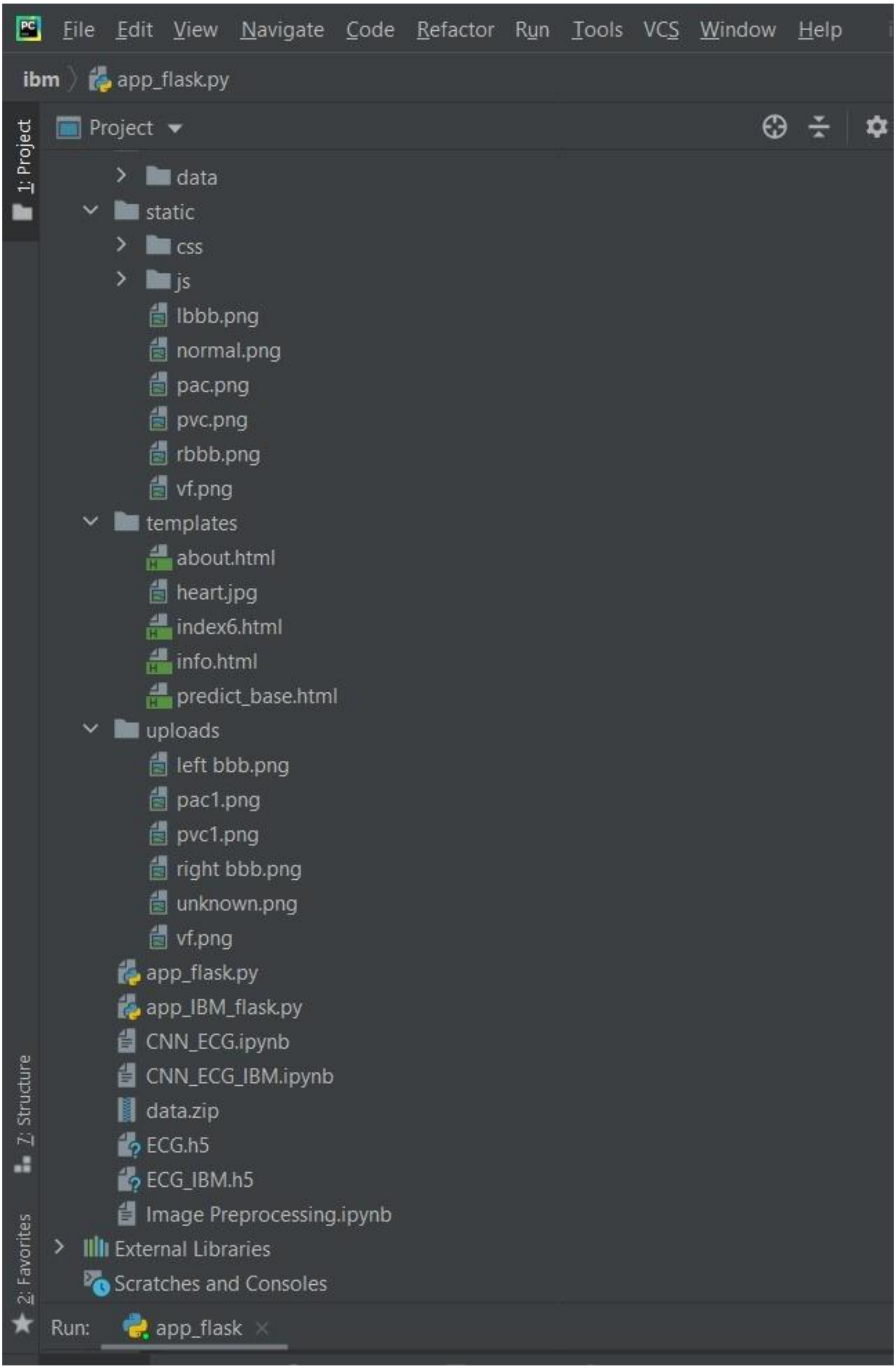
information.html displays all important details to be known about ECG.



1. predict-base.html and predict.html acceptinput from the user and predicts the values.


Home Info Predict
ECG Arrhythmia Classification
Choose...
Predict!


Home Info Predict
ECG Arrhythmia Classification
Choose...
Result: Normal



1. **Building server-side script:**

We will build the flask file ‘app.py’ which is a web frameworkwritten in pythonfor server-side scripting.

1. The app starts running when the “_name_”constructor is called in main.
2. render_template is used to return HTML file.
3. “GET” method is used to take input from the user.
4. “POST” method is used to display the output to the user.

```python
import os
import numpy as np #used for numerical analysis
from flask import Flask,request,render_template
# Flask-It is our framework which we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
#render_template- used for rendering the html pages
from tensorflow.keras.models import load_model#to load our trained model
from tensorflow.keras.preprocessing import image

app=Flask(__name__)#our flask app
model=load_model('ECG.h5')#loading the model

@app.route("/") #default route
def about():
    return render_template("home.html")#rendering html page

@app.route("/about") #default route
def home():
    return render_template("home.html")#rendering html page

@app.route("/info") #default route
def information():
    return render_template("information.html")#rendering html page

@app.route("/upload") #default route
def test():
    return render_template("predict.html")#rendering html page
```

```python
def upload():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        basepath=os.path.dirname('__file__')#storing the file directory
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)#converting image to array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred=model.predict(x)#predicting classes
        y_pred = np.argmax(pred)
        print("prediction",y_pred)#printing the prediction

        index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction',
       'Premature Ventricular Contractions', 'Right Bundle Branch Block','Ventricular Fibrillation']
        result=str(index[y_pred])

        return result#resturing the result
    return None

#port = int(os.getenv("PORT"))
if __name__=="__main__":
    app.run(debug=False)#running our app
    #app.run(host='0.0.0.0', port=8000)
```

Running The App:

```
C:\Users\M Sheshikiran Reddy\VIT\20BAI1061\CNN_PROJECT_SMARTINTERNZ\flask>python app_flask.py
```

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app_flask' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

Training model in IBM WATSON STUDIO

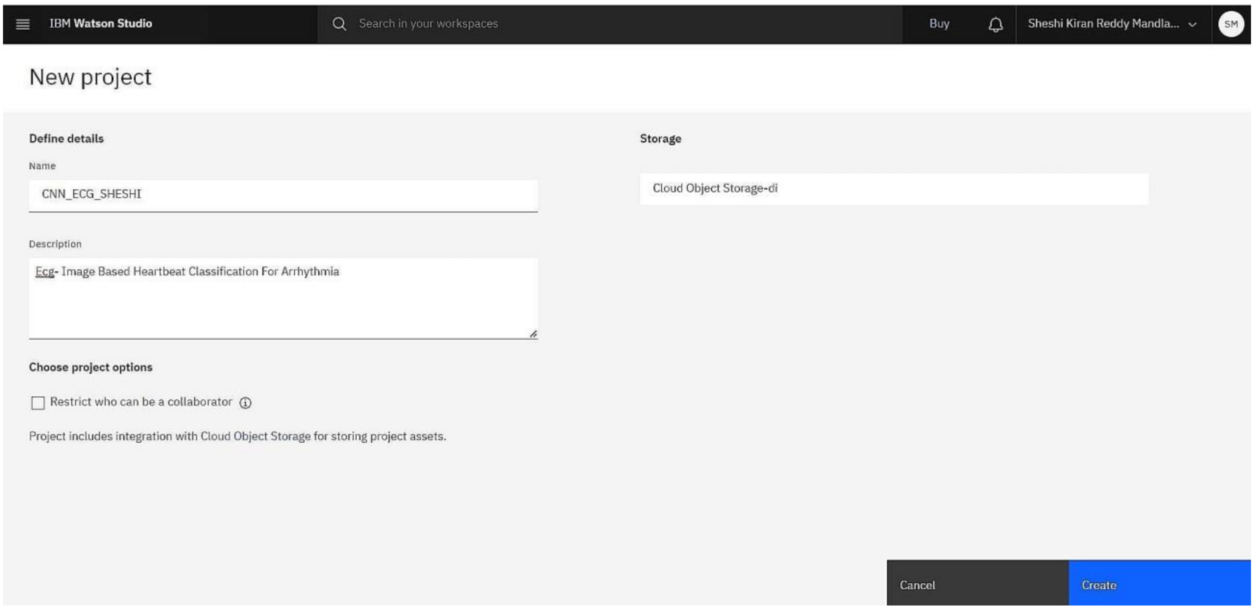
1.Creating IBM cloud account:

We have to create an IBM Cloud Account and should log in.

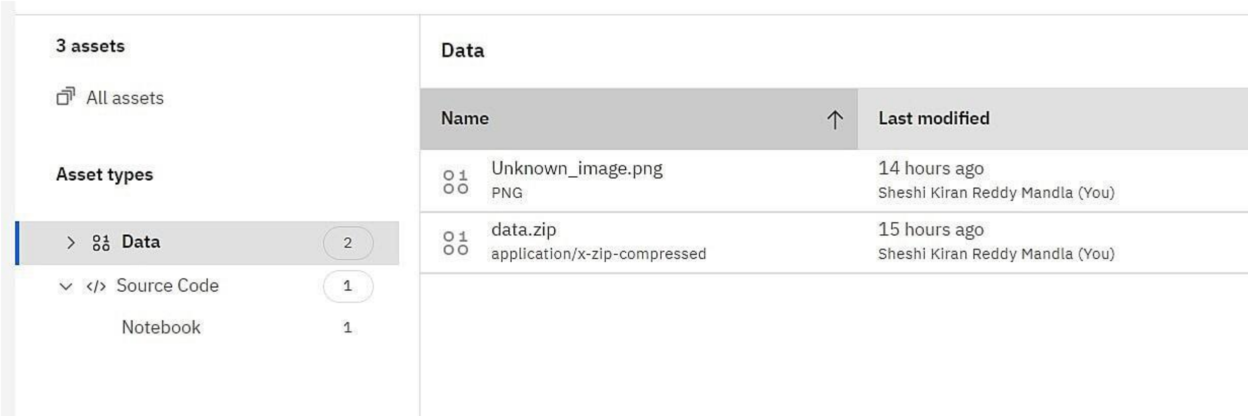
2.Creating Watson StudioService & MachineLearning Service:

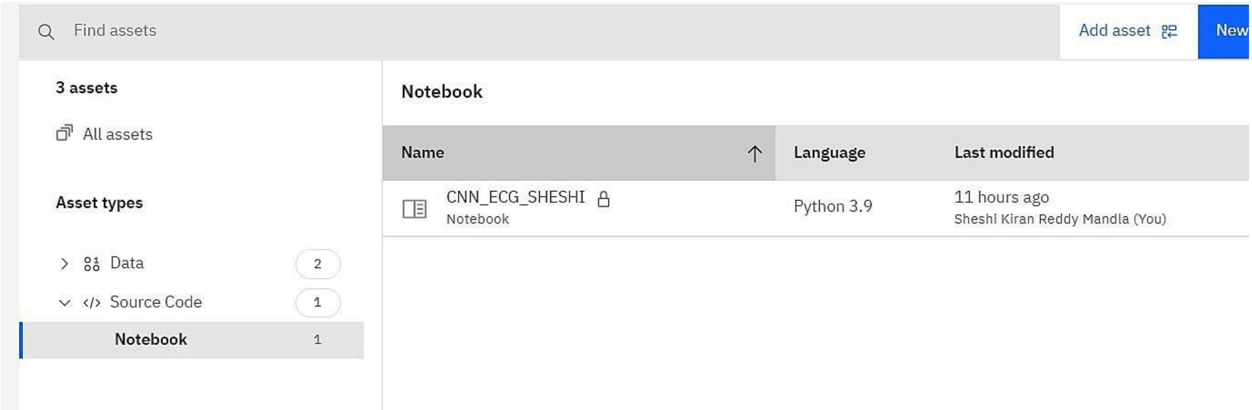
Machine Learning-7n	Default	Dallas	Machine Learning	Active
Watson Studio-yy	Default	Dallas	Watson Studio	Active

3.Create a Project& Deployment spacein the watson studio:



4.Upload The datasetand create a jupytersource file in the created project:





5.Apply CNN algorithmand save the model and deploy it using API key generated:

```
In [77]: model.save('ECG_IBM.h5')
```

```
In [109]: !tar -zcvf ECG-arrhythmia-classification-model_new.tgz ECG_IBM.h5
```

```
ECG_IBM.h5
```

```
In [110]: ls -1
```

```
data/
ECG-arrhythmia-classification-model_new.tgz
ECG-classification.tgz
ECG_IBM.h5
```

```
In [112]: from ibm_watson_machine_learning import APIClient
          wml_credentials={
              "url":"https://us-south.ml.cloud.ibm.com",
              "apikey":"EfnNlIAqu-_QXB0QsQqQQlnwkes_B9ssggk8ipjZQH67"
          }
          client=APIClient(wml_credentials)

In [121]: client.spaces.list()

          Note: 'limit' is not provided. Only first 50 records will be displayed if the number of records exceed 50
          ------------------------------------  ------------------  ------------------------
          ID                                    NAME                CREATED
          fc75767f-659b-4dc3-a238-cbb53d201cf2  CNN_ECG_IBM_SHESHI  2022-07-05T10:52:44.075Z
          ------------------------------------  ------------------  ------------------------

In [122]: space_uid="fc75767f-659b-4dc3-a238-cbb53d201cf2"

In [123]: client.set.default_space(space_uid)

Out[123]: 'SUCCESS'

In [124]: client.set.default_space(space_uid)

Out[124]: 'SUCCESS'

In [126]: software_spec_uid = client.software_specifications.get_uid_by_name("tensorflow_rt22.1-py3.9")
          software_spec_uid

Out[126]: 'acd9c798-6974-5d2f-a657-ce06e986df4d'

In [143]: model_details = client.repository.store_model(model='ECG-arrhythmia-classification-model_new.tgz',meta_props={
              client.repository.ModelMetaNames.NAME:"CNN_SHESHI",
              client.repository.ModelMetaNames.TYPE:"tensorflow_2.7",
              client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid})
          model_id=client.repository.get_model_uid(model_details)

          This method is deprecated, please use get_model_id()

In [144]: model_id

Out[144]: '70742fe8-7ac8-4855-994c-b572931ef787'

In [147]: client.repository.download(model_id,'my_model.tar1.gz')

          Successfully saved model content to file: 'my_model.tar1.gz'

Out[147]: '/home/wsuser/work/my_model.tar1.gz'
```

6.For downloading the model we have to run the last part of the above code in the local jupyternotebook:

```
In [ ]:  1 client.repository.download('70742fe8-7ac8-4855-994c-b572931ef787','my_model_sheshi.tar.gz')
```

7.Now we will extractthe .h5 model�le and will do the app deployment using �ask as done in the previoustraining

```python
import os
import numpy as np  # used for numerical analysis
from flask import Flask, request, render_template
# Flask-It is our framework which we are going to use to run/serve our appl
# request-for accessing file which was uploaded by the user on our applicat
# render_template- used for rendering the html pages
from tensorflow.keras.models import load_model  # to load our trained model
from tensorflow.keras.preprocessing import image

app = Flask(__name__)  # our flask app
model = load_model('ECG_IBM.h5')  # loading the model


@app.route("/")  # default route
def about():
    return render_template("home.html")  # rendering html page
```

Hence we trainedthe model using IBM Watson