

CLASSIFICATION OF ARRHYTHMIA BY USING DEEP LEARNING WITH 2-D ECG SPECTRAL IMAGE REPRESENTATION

INTRODUCTION:

1.1 Overview:

According to the World Health Organization (WHO), cardiovascular diseases (CVDs) are the number one cause of death today. Over 17.7 million people died from CVDs in the year 2017 all over the world which is about 31% of all deaths, and over 75% of these deaths occur in low and middle-income countries. Arrhythmia is a representative type of CVD that refers to any irregular change from the normal heart rhythms. There are several types of arrhythmia including atrial fibrillation, premature contraction, ventricular fibrillation, and tachycardia. Although a single arrhythmia heartbeat may not have a serious impact on life, continuous arrhythmia beats can result in fatal circumstances. In this project, we build an effective electrocardiogram (ECG) arrhythmia classification method using a convolutional neural network (CNN), in which we classify ECG into seven categories, one being normal and the other six being different types of arrhythmia using deep two-dimensional CNN with grayscale ECG images. We are creating a web application where the user selects the image which is to be classified. The image is fed into the model that is trained and the cited class will be displayed on the webpage.

1.2 Purpose:

In the past few decades, Deep Learning has proved to be a compelling tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks. In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

LITERATURE SURVEY:

2.1 Existing Problem:

Cardiovascular diseases (CVDs) are the number one cause of death today. Over 17.7 million people died from CVDs in the year 2017 all over the world which is about 31% of all deaths, and over 75% of these deaths occur in low and middle-income countries. Arrhythmia is a representative type of CVD that refers to any irregular change from the

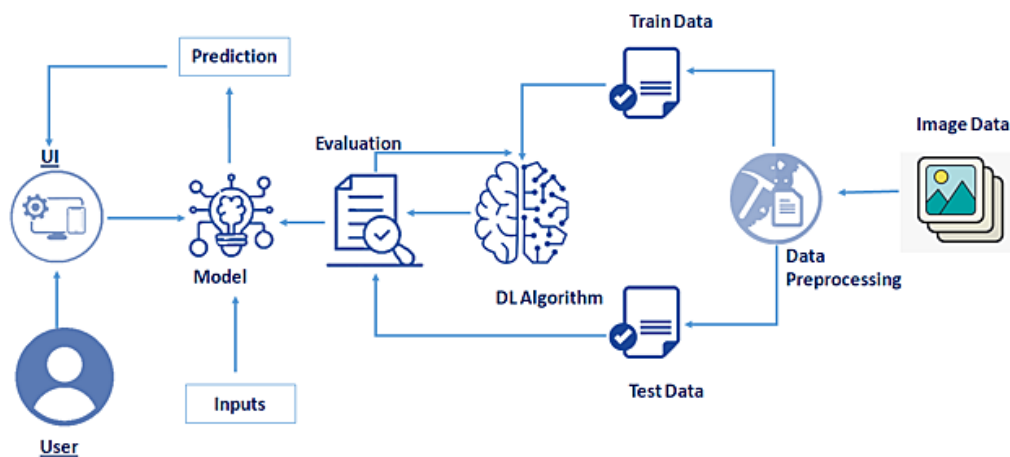
normal heart rhythms. There are several types of arrhythmia including atrial fibrillation, premature contraction, ventricular fibrillation, and tachycardia.

2.2 Proposed Solution:

An "ambulatory electrocardiogram" or an ECG about the size of a postcard or digital camera that the patient will be using for 1 to 2 days, or up to 2 weeks. The test measures the movement of electrical signals or waves through the heart. These signals tell the heart to contract (squeeze) and pump blood. The patient will have electrodes taped to your skin. It's painless, although some people have mild skin irritation from the tape used to attach the electrodes to the chest. They can do everything but shower or bathe while wearing the electrodes. After the test period, patient will go back to see your doctor. They will be downloading the information.

THEORETICAL EXPERIENCE:

3.1 Block Diagram:



We will prepare the project by following the below steps:

- We will be working with Sequential type of modeling
- We will be working with Keras capabilities
- We will be working with image processing techniques
- We will build a web application using the Flask framework.
- Afterwards we will be training our dataset in the IBM cloud and building another model from IBM and we will also test it.

3.2 Hardware & Software Desgining:

Hardware Components used:

Since we are using the IBM cloud as a platform to execute this project we don't need any hardware components other than our system.

Software Components Used:

Anaconda Navigator:

We will be using Anaconda Navigator which is installed in our system and Watson studio from the IBM cloud to complete the project. Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, crossplatform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and spyder

Watson Studio:

Watson Studio is one of the core services in Cloud Pak for Data as a Service. Watson Studio provides you with the environment and tools to solve your business problems by collaboratively working with data. You can choose the tools you need to analyze and visualize data, to cleanse and shape data, or to build machine learning models.

This illustration shows how the architecture of Watson Studio is centered around the project. A project is a workspace where you organize your resources and work with data.

Watson Studio projects fully integrate with the catalogs and deployment spaces:

- Deployment spaces are provided by the Watson Machine Learning service
- You can easily move assets between projects and deployment spaces.

Experimental Investigations:

In this project, we have deployed our training model using CNN on IBM Watson studio and in our local machine. We are deploying 4 types of CNN layers in a sequential manner , starting from :

1. Convolutional layer 2D:

A 2-D convolutional layer applies sliding convolutional filters to 2-D input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then adding a bias term.

2. Pooling Layer :

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarises the features present in a region of the feature map generated by a convolution layer.

3. Fully-Connected layer :

After extracting features from multiple convolution layers and pooling layers, the fully-connected layer is used to expand the connection of all features. Finally, the SoftMax layer makes a logistic regression classification. Fully-connected layer transfers the weighted sum of the output of the previous layer to the activation function.

4. Dropout Layer :

There is usually a dropout layer before the fullyconnected layer. The dropout layer will temporarily disconnect some neurons from the network according to the certain probability during the training of the convolution neural network, which reduces the joint adaptability between neuron nodes, reduces overfitting, and enhances the generalization ability of the network.

PROCEDURE:

5.1 Flow Chart & Results by training model in local machine:

a. Dataset Collection:

The dataset contains six classes:

1. Left Bundle Branch Block
2. Normal
3. Premature Atrial Contraction
4. Premature Ventricular Contractions
5. Right Bundle Branch Block
6. Ventricular Fibrillation

b. Image Preprocessing:

Image Pre-processing includes the following main tasks:

- Import ImageDataGenerator Library:

Image data augmentation is a technique that 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 neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

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

- Configure ImageDataGenerator Class:

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

- Image shifts via the width_shift_range and height_shift_range arguments.
- Image flips via the horizontal_flip and vertical_flip arguments.
- Image rotates via the rotation_range argument
- Image brightness via the brightness_range argument.
- Image zooms via the zoom_range argument.

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

```
[6]: #setting parameter for image data augmentation to the training data
train_datagen=ImageDataGenerator(rescale=1./225,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
#image data augmentation to test dataset
test_datagen=ImageDataGenerator(rescale=1./225)
```

- Applying ImageDataGenerator functionality to the trainset and test set:

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

```
In [7]: #performing data agumentation to train the dataset
x_train=train_datagen.flow_from_directory(directory='C:/Users/Saipriya/OneDrive/Desktop/data/train',target_size=(64,64),batch_size=32,class_mode='cate
#performing agumentation to test the dataset
x_test=test_datagen.flow_from_directory(directory='C:/Users/Saipriya/OneDrive/Desktop/data/test',target_size=(64,64),batch_size=32,class_mode='categor

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

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. 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.

c. 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:

- Import the model building Libraries:

```
In [8]: import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.layers import Conv2D,MaxPooling2D
import keras
```

- Initializing the model:

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, 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.

```
In [9]: model= keras.Sequential()
```

- Adding CNN Layers:

We are adding a convolution layer with an activation function as “relu” and with a small filter size (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 [10]: #adding model layer
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
```

- Adding Output Layers:

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

```
In [11]: model.add(Dense(32))
model.add(Dense(6,activation='softmax'))

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 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 32)	200736
dense_1 (Dense)	(None, 6)	198
=====		
Total params: 211,078		
Trainable params: 211,078		
Non-trainable params: 0		

- Configure the Learning Process:

- 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.

- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in the training process.

```
In [12]: model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

● Training the model:

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

```
In [13]: model.fit_generator(generator=x_train,steps_per_epoch = len(x_train), epochs=10, validation_data=x_test,validation_steps = len(x_test))
```

C:\Users\Saipriya\AppData\Local\Temp\ipykernel_16540\53529216.py:1: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators.

```
model.fit_generator(generator=x_train,steps_per_epoch = len(x_train), epochs=10, validation_data=x_test,validation_steps = len(x_test))
```

Epoch 1/10
480/480 [=====] - 96s 184ms/step - loss: 0.7718 - accuracy: 0.7376 - val_loss: 0.4899 - val_accuracy: 0.8243
Epoch 2/10
480/480 [=====] - 103s 225ms/step - loss: 0.2681 - accuracy: 0.9218 - val_loss: 0.4358 - val_accuracy: 0.8598
Epoch 3/10
480/480 [=====] - 93s 194ms/step - loss: 0.2177 - accuracy: 0.9355 - val_loss: 0.3285 - val_accuracy: 0.9116
Epoch 4/10
480/480 [=====] - 84s 175ms/step - loss: 0.1840 - accuracy: 0.9459 - val_loss: 0.3467 - val_accuracy: 0.8919
Epoch 5/10
480/480 [=====] - 101s 210ms/step - loss: 0.1577 - accuracy: 0.9529 - val_loss: 0.3456 - val_accuracy: 0.9034
Epoch 6/10
480/480 [=====] - 96s 200ms/step - loss: 0.1411 - accuracy: 0.9576 - val_loss: 0.4097 - val_accuracy: 0.8983
Epoch 7/10
480/480 [=====] - 83s 173ms/step - loss: 0.1204 - accuracy: 0.9621 - val_loss: 0.4474 - val_accuracy: 0.8908
Epoch 8/10
480/480 [=====] - 79s 165ms/step - loss: 0.1196 - accuracy: 0.9643 - val_loss: 0.3226 - val_accuracy: 0.9197
Epoch 9/10
480/480 [=====] - 66s 126ms/step - loss: 0.1058 - accuracy: 0.9685 - val_loss: 0.4323 - val_accuracy: 0.8963
Epoch 10/10
480/480 [=====] - 68s 141ms/step - loss: 0.1028 - accuracy: 0.9681 - val_loss: 0.3876 - val_accuracy: 0.9127

● Saving the model:

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 [14]: model.save('ECG.h5')
```

● Testing the model:

Load necessary libraries and load the saved model using load_model

Taking an image as input and 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 [15]: from tensorflow.keras.models import load_model
         from tensorflow.keras.preprocessing import image
         model=load_model("ECG.h5")

In [16]: img = image.load_img("C:/Users/Saipriya/OneDrive/Desktop/data/test/Premature Atrial Contraction/fig_26.png",target_size=(64,64))
         x = image.img_to_array(img)
         x = np.expand_dims(x,axis=0)
         pred = model.predict(x)
         y_pred = np.argmax(pred)
         y_pred

1/1 [=====] - 3s 3s/step

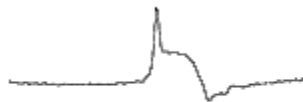
Out[16]: 2

In [17]: index=['Left Bundle Branch block','Normal','Premature Atrial Contraction','Premature Ventricular Contraction','Right Bundle Branch Block','Ventricular
         result = str(index[y_pred])
         result

Out[17]: 'Premature Atrial Contraction'

```

The unknown image uploaded is:



Here the output for the uploaded result is Premature Atrial Contraction

d. 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 section has the following tasks

- Building HTML Pages:
 - We use HTML to create the front end part of the web page.
 - Here, we created 4 html pages- home.html, predict_base.html, predict.html, information.html.
 - home.html displays the home page.
 - information.html displays all important details to be known about ECG.
 - predict-base.html and predict.html accept input from the user and predicts the values.

Welcome To ECG- Image Based Heartbeat Classification Application For Arrhythmia Detection



NORMAL

Note that the heart is beating in a regular sinus rhythm between 60 - 100 beats per minute (specifically 82 bpm). All the important intervals on this recording are within normal ranges.

The normal ECG patterns seen in children differ considerably from those in adults.



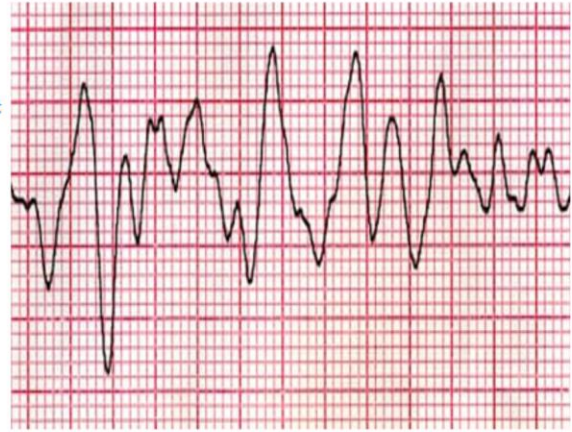
VENTRICULAR FIBRILLATION

A life-threatening heart rhythm that results in a rapid, inadequate heartbeat.

Ventricular fibrillation (VF) is a rapid, life-threatening heart rhythm starting in the bottom chambers of the heart. It can be triggered by a heart attack,

Because the heart doesn't pump adequately during ventricular fibrillation, sustained VF can cause low blood pressure, loss of consciousness or death.

Emergency treatment includes immediate defibrillation with an automated external defibrillator (AED) and cardiopulmonary resuscitation (CPR). Long-term therapy includes implantable defibrillators and medications to prevent recurrence.



ECG Arrhythmia Classification



ECG Arrhythmia Classification



ECG Arrhythmia Classification



Result: Ventricular Fibrillation

- Building server-side script:

We will build the flask file 'app.py' which is a web framework written in python for server-side scripting.

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

Name	Date Modified
<> Info.html	05-11-2022 11:10
<> main.css	05-11-2022 11:38
main.js	05-11-2022 11:40
uploads	12-11-2022 13:13
lbbbp.png	12-11-2022 11:53
normalpd.png	05-11-2022 13:57
pacpd.png	02-11-2022 11:45
pvcpd.png	12-11-2022 11:51
rbbbp.png	05-11-2022 13:58
VFEfig_138.png	12-11-2022 11:52
app_IBM.py	12-11-2022 11:37
app.py	05-11-2022 12:00
ECG_IBM.h5	12-11-2022 11:39
ECG.h5	04-11-2022 12:14

Spyder (Python 3.9)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Saipriya\OneDrive\Desktop\PROJECT DEVELOPMENT PHASE\SPRINT 4\CLOUD DEPLOYMENT\Flask\app.py

untitled1.py X app.py X

1  import os
2  import numpy as np #used for numerical analysis
3  from flask import Flask,request,render_template
4  # Flask-It is our framework which we are going to use to run/serve our application.
5  #request-for accessing file which was uploaded by the user on our application.
6  #render_template- used for rendering the html pages
7  from tensorflow.keras.models import load_model#to load our trained model
8  from tensorflow.keras.preprocessing import image
9
10 app=Flask(__name__)#our flask app
11 model=load_model('ECG.h5')#loading the model
12
13 @app.route("/") #default route
14 def about():
15     return render_template("about.html")#rendering html page
16
17 @app.route("/about") #default route
18 def home():
19     return render_template("about.html")#rendering html page
20
21 @app.route("/info") #default route
22 def information():
23     return render_template("info.html")#rendering html page
24
25 @app.route("/upload") #default route
26 def test():
27     return render_template("index6.html")#rendering html page
28
29

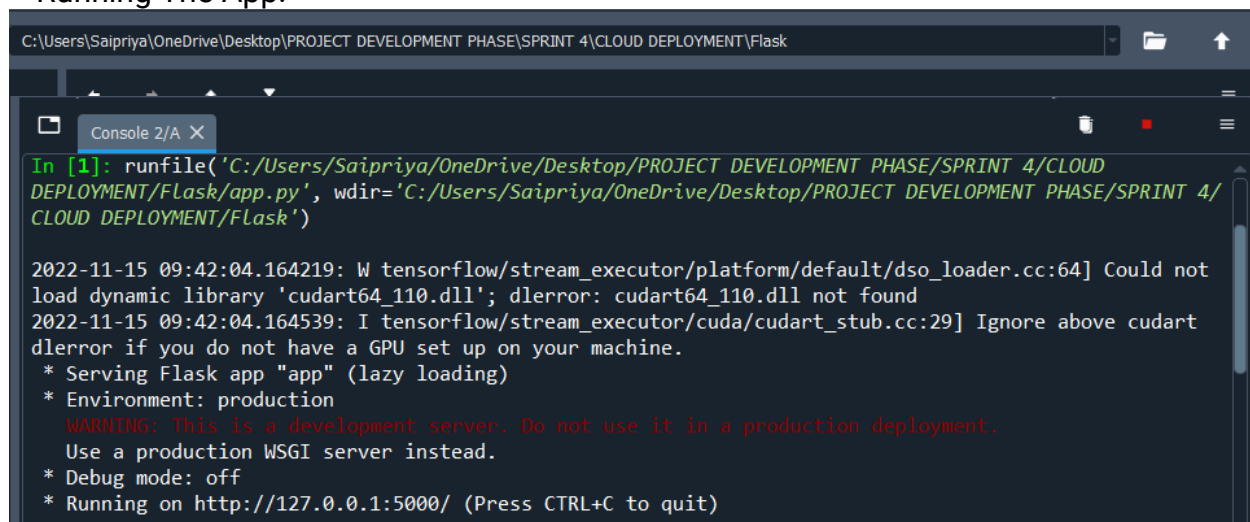
```

```

30 @app.route("/predict",methods=["GET","POST"]) #route for our prediction
31 def upload():
32     if request.method=='POST':
33         f=request.files['file'] #requesting the file
34         basepath=os.path.dirname('__file__')#storing the file directory
35         filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
36         f.save(filepath)#saving the file
37
38         img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
39         x=image.img_to_array(img)#converting image to array
40         x=np.expand_dims(x,axis=0)#changing the dimensions of the image
41
42         pred=model.predict(x)#predicting classes
43         y_pred = np.argmax(pred)
44         print("prediction",y_pred)#printing the prediction
45
46         index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction',
47               'Premature Ventricular Contractions', 'Right Bundle Branch Block','Ventricular Fibrillation']
48         result=str(index[y_pred])
49
50         return result#returning the result
51     return None
52
53 #port = int(os.getenv("PORT"))
54 if __name__=="__main__":
55     app.run(debug=False)#running our app
56     #app.run(host='0.0.0.0', port=8000)

```

● Running The App:



```

C:\Users\Saipriya\OneDrive\Desktop\PROJECT DEVELOPMENT PHASE\SPRINT 4\CLOUD DEPLOYMENT\Flask
In [1]: runfile('C:/Users/Saipriya/OneDrive/Desktop/PROJECT DEVELOPMENT PHASE/SPRINT 4/CLOUD DEPLOYMENT/Flask/app.py', wdir='C:/Users/Saipriya/OneDrive/Desktop/PROJECT DEVELOPMENT PHASE/SPRINT 4/CLOUD DEPLOYMENT/Flask')

2022-11-15 09:42:04.164219: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found
2022-11-15 09:42:04.164539: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
* Serving Flask app "app" (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.

5.2 TRAINING MODEL IN IBM WATSON STUDIO:

a. Creating IBM cloud account:

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

b. Creating Watson Studio Service & Machine Learning Service:

Storage (1)						
	Cloud Object Storage-fj	Default	Global	Cloud Object Storage	Active	cpdaas
AI / Machine Learning (2)						
	Watson Machine Learning-rn	Default	Dallas	Watson Machine Learning	Active	cpdaas
	Watson Studio-dd	Default	Dallas	Watson Studio	Active	—

c. Create a Project & Deployment space in the watson studio:

IBM Watson Studio

Search in your workspaces

Buy

KALAKRETI KAMALAKKAN...

Dallas

KK

New project

Define details

Name

Description

Choose project options

☐ Restrict who can be a collaborator ⓘ

☐ Mark as sensitive ⓘ

Project includes integration with [Cloud Object Storage](#) for storing project assets.

Storage

Cancel

Create

IBM Watson Studio

Search in your workspaces

Buy

KALAKRETI KAMALAKKAN...

Dallas

KK

Deployments /

Classification of Arrhythmia

Overview


Assets

Deployments

Jobs

Manage

Assets

 CNN
4 days ago

[View all \(1\)](#)

Deployments

All

1

0

Deployed

Failed

[View deployments](#)

Job runs

0


0


Active

Failed last 24 hours

[View jobs](#)

Space activity

 **Online deployment ready**
The online deployment **CNN** in space **Classification of Arrhythmia** is ready to accept requests
Today at 10:38 AM

 **Online deployment created**
You created online deployment **"CNN"** in space **Classification of Arrhythmia**. You must wait for the deployment to enter ready state before submitting requests.
Today at 10:38 AM

d. Upload The dataset and create a jupyter source file in the created project:

IBM Watson Studio

Search in your workspaces

Buy

KALAKRETI KAMALAKKAN...

Dallas

Projects / Classification of Arrhythmia

Overview Assets Jobs Manage

Find assets

Import assets

New asset

2 assets

All assets

Asset types

Data 1

Notebooks 1

Name	Last modified
Model Building Notebook	4 days ago Modified by you
Classification of Arrhythmia by Using Deep Learning w... ZIP	5 days ago Modified by you

Items per page: 20

1-2 of 2 items

1 of 1 pages

Data in this project

Drop data files here or browse for files to upload

IBM Watson Studio

Search in your workspaces

Buy

KALAKRETI KAMALAKKAN...

Dallas

Projects / Classification of Arrhythmia

Overview Assets Jobs Manage

Find assets

Import assets

New asset

2 assets

All assets

Asset types

Data 1

Notebooks 1

Name	Language	Last modified
Model Building Notebook	Python 3.9	4 days ago Modified by you

Items per page: 20

1-1 of 1 items

1 of 1 pages

Data in this project

Drop data files here or browse for files to upload

e. Apply CNN algorithm and save the model and deploy it using API key generated:

```
In [15]: model.save('ECG_IBM.h5')

In [16]: !tar -zcvf ECG-arrhythmia-classification-model_new.tgz ECG_IBM.h5

In [17]: ls -l
data/
ECG-arrhythmia-classification-model_new.tgz
ECG_IBM.h5
```

```

In [19]: from ibm_watson_machine_learning import APIClient
wml_credentials={
    "url":"https://us-south.ml.cloud.ibm.com",
    "apikey":"rrkT3wpDakBr3E-Rd6iN2M2wmdMQj7Qv1-TnlziN36y"
}
client=APIClient(wml_credentials)

In [20]: def guid_from_space_name(client,space_name):
space=client.spaces.get_details()
#print(space)
return(next(item for item in space ['resources'] if item['entity']['name']==space_name)['metadata']['id'])

In [21]: space_uid=guid_from_space_name(client,'Classification of Arrhythmia')
print("Space UID= "+space_uid)
Space UID= 7277f3b7-fd81-4943-a294-8bca90dab1f6

In [22]: client.set.default_space(space_uid)

Out[22]: 'SUCCESS'

In [24]: software_spec_uid = client.software_specifications.get_uid_by_name("tensorflow_rt22.1-py3.9")
software_spec_uid

Out[24]: 'acd9c798-6974-5d2f-a657-ce06e986df4d'

In [25]: model_details = client.repository.store_model(model='ECG-arrhythmia-classification-model_new.tgz',meta_props={
client.repository.ModelMetadata.NAME:"CMH",
client.repository.ModelMetadata.TYPE:"tensorflow 2.7",
client.repository.ModelMetadata.SOFTWARE_SPEC_UID:software_spec_uid})
model_id=client.repository.get_model_uid(model_details)

This method is deprecated, please use get_model_id()
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages/ibm_watson_machine_learning/repository.py:1453: UserWarning: This method is deprecated, please use get_model_id()
warn("This method is deprecated, please use get_model_id()")

In [26]: model_id

Out[26]: '674dcec5-ca13-495d-b9a5-2a5af1e68e9d'

In [27]: client.repository.download(model_id,'my_model.tar1.gz')

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

Out[27]: '/home/username/work/my_model.tar1.gz'

Test the Model

In [33]: from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model=load_model("ECG_IBM.h5")

In [34]: img = image.load_img("/home/username/work/data/test/Premature Atrial Contraction/fig_26.png",target_size=(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
pred = model.predict(x)
y_pred = np.argmax(pred)
y_pred

Out[34]: 2

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]: 'Premature Atrial Contraction'

```

f. For downloading the model we have to run the last part of the above code in the local jupyter notebook:

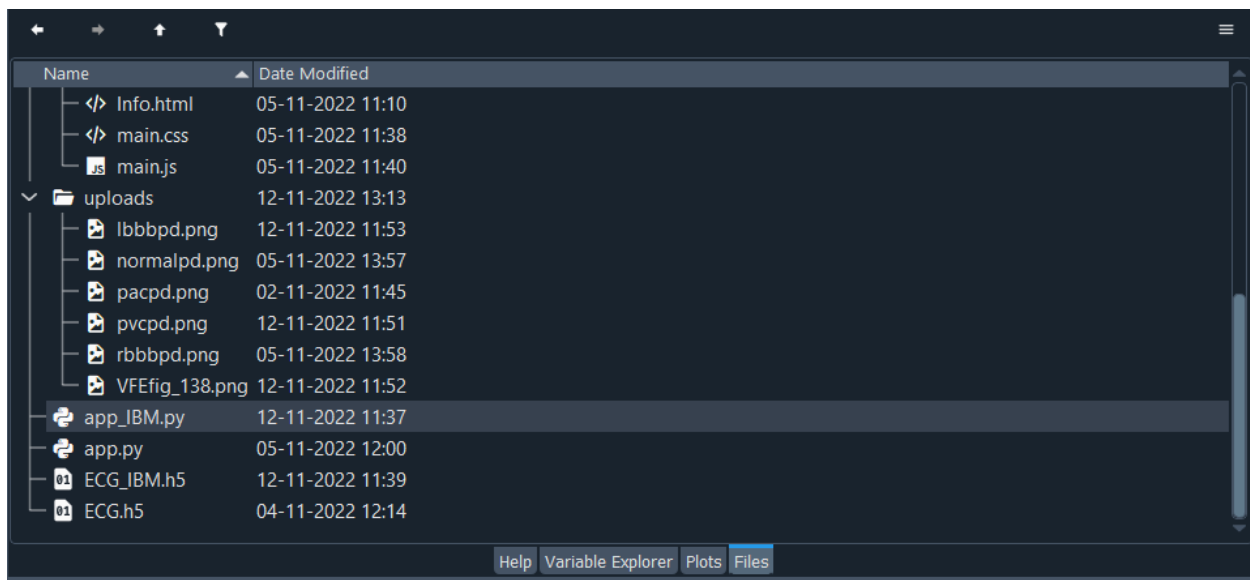
```

In [12]: client.repository.download('674dcec5-ca13-495d-b9a5-2a5af1e68e9d','my_model.tar1.gz')

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

Out[12]: 'C:\Users\Sneha\my_model.tar1.gz'

```

g. Now we will extract the .h5 model file and will do the app deployment using flask as done in the previous training:

```
...Users\Saipriya\OneDrive\Desktop\PROJECT DEVELOPMENT PHASE\SPRINT 4\CLOUD DEPLOYMENT\Flask\app_IBM.py

untitled1.py X app_IBM.py X

1  import os
2  import numpy as np #used for numerical analysis
3  from flask import Flask,request,render_template
4  # Flask-It is our framework which we are going to use to run/serve our application.
5  #request-for accessing file which was uploaded by the user on our application.
6  #render_template- used for rendering the html pages
7  from tensorflow.keras.models import load_model#to load our trained model
8  from tensorflow.keras.preprocessing import image
9
10 app=Flask(__name__)#our flask app
11 model=load_model('ECG_IBM.h5')#loading the model
12
13 @app.route("/") #default route
14 def about():
15     return render_template("about.html")#rendering html page
16
17 @app.route("/about") #default route
18 def home():
19     return render_template("about.html")#rendering html page
20
21 @app.route("/info") #default route
22 def information():
23     return render_template("info.html")#rendering html page
24
25 @app.route("/upload") #default route
26 def test():
27     return render_template("index6.html")#rendering html page
28
```

```

31 def upload():
32     if request.method=='POST':
33         f=request.files['file'] #requesting the file
34         basepath=os.path.dirname('__file__')#storing the file directory
35         filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
36         f.save(filepath)#saving the file
37
38         img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
39         x=image.img_to_array(img)#converting image to array
40         x=np.expand_dims(x,axis=0)#changing the dimensions of the image
41
42         pred=model.predict(x)#predicting classes
43         y_pred = np.argmax(pred)
44         print("prediction",y_pred)#printing the prediction
45
46         index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction',
47               'Premature Ventricular Contractions', 'Right Bundle Branch Block','Ventricular Fibrillation']
48         result=str(index[y_pred])
49
50         return result#returning the result
51     return None
52
53 #port = int(os.getenv("PORT"))
54 if __name__=="__main__":
55     app.run(debug=False)#running our app
56     #app.run(host='0.0.0.0', port=8000)

```

Hence we trained the model using IBM Watson.

Advantages & Disadvantages:

6.1 Advantages:

- The proposed model predicts Arrhythmia in images with a high accuracy rate of nearly 96%
- The early detection of Arrhythmia gives better understanding of disease causes, initiates therapeutic interventions and enables developing appropriate treatments.

6.2 Disadvantages:

- Not useful for identifying the different stages of Arrhythmia disease.
- Not useful in monitoring motor symptoms

Applications :

- It is useful for identifying the arrhythmia disease at an early stage.
- It is useful in detecting cardiovascular disorders

Conclusion:

- Cardiovascular disease is a major health problem in today's world. The early diagnosis of cardiac arrhythmia highly relies on the ECG.
- Unfortunately, the expert level of medical resources is rare, visually identify the ECG signal is challenging and time-consuming.
- The advantages of the proposed CNN network have been put to evidence.

- It is endowed with an ability to effectively process the non-filtered dataset with its potential anti-noise features. Besides that, ten-fold cross-validation is implemented in this work to further demonstrate the robustness of the network.

Future Scope:

For future work, it would be interesting to explore the use of optimization techniques to find a feasible design and solution. The limitation of our study is that we have yet to apply any optimization techniques to optimize the model parameters and we believe that with the implementation of the optimization, it will be able to further elevate the performance of the proposed solution to the next level.

References:

- <https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution2dlayer.html;jsessionid=0a7e3bc26fabda07a5032030294b>
- <https://github.com/Anshuman151/ECG-Image-Based-Heartbeat-Classification-for-Arrhythmia-Detection-Using-IBM-Watson-Studio/blob/main/README.md>
- <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>