

Sprint-2

MODEL BUILDING

Date	17 Nov 2022
Team ID	PNT2022TMID30834
Project Name	Classification of Arrhythmia by Using Deep Learning with 2-D ECG Spectral Image Representation

Task

1. 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 Libraries:

```
[ ] 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

[ ] from tensorflow.keras.datasets import mnist
```

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

Adding CNN Layer:

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.

```
model=Sequential()
```

Add Layers (Convolution,MaxPooling,Flatten,Dense-(Hidden Layers),Output)

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

```
[ ] model.add(MaxPooling2D(pool_size=(2,2)))
```

```
[ ] model.add(Flatten())
```

```
[ ] model.add(Dense(300,activation='relu'))
```

```
[ ] model.add(Dense(5,activation="softmax"))
```

Adding Dense Layer:

Dense layer is deeply connected neural network layer. It is most common and frequently used layer.

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

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

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

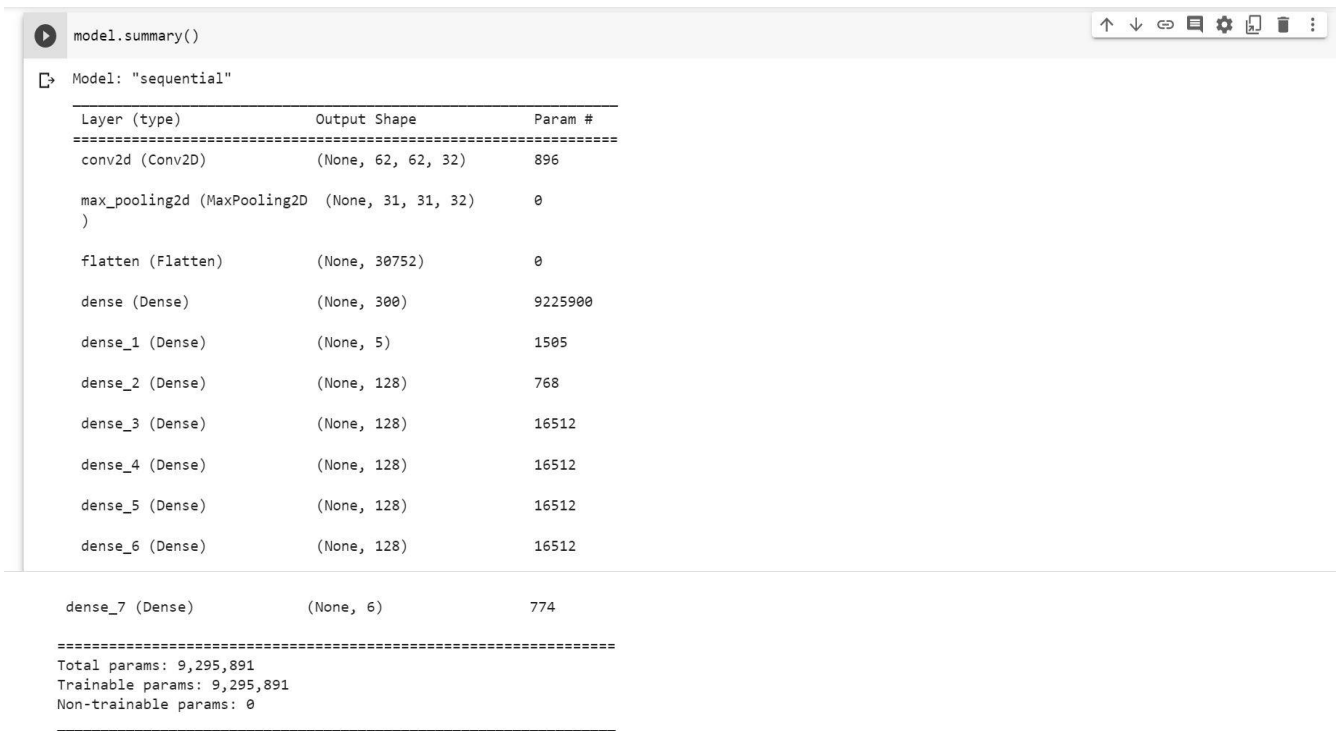
```
[ ] model.add(Dense(units=128,kernel_initializer="random_uniform",activation="relu"))
```

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

Adding Output Layer:

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

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



```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
flatten (Flatten)	(None, 30752)	0
dense (Dense)	(None, 300)	9225900
dense_1 (Dense)	(None, 5)	1505
dense_2 (Dense)	(None, 128)	768
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 128)	16512
dense_5 (Dense)	(None, 128)	16512
dense_6 (Dense)	(None, 128)	16512
dense_7 (Dense)	(None, 6)	774

=====
Total params: 9,295,891
Trainable params: 9,295,891
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 processes

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.

Compile The Model

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

Train The Model:

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

```
model.fit(x_train, epochs=9, validation_data=x_test, steps_per_epoch=len(x_train), validation_steps=len(x_test))
```

```
Epoch 1/9
285/285 [=====] - 105s 369ms/step - loss: 1.5908 - accuracy: 0.3193 - val_loss: 1.5114 - val_accuracy: 0.4788
Epoch 2/9
285/285 [=====] - 103s 360ms/step - loss: 1.5822 - accuracy: 0.3193 - val_loss: 1.5157 - val_accuracy: 0.4788
Epoch 3/9
285/285 [=====] - 102s 359ms/step - loss: 1.5828 - accuracy: 0.3193 - val_loss: 1.4901 - val_accuracy: 0.4788
Epoch 4/9
285/285 [=====] - 104s 365ms/step - loss: 1.5826 - accuracy: 0.3193 - val_loss: 1.5292 - val_accuracy: 0.4788
Epoch 5/9
285/285 [=====] - 105s 370ms/step - loss: 1.5820 - accuracy: 0.3193 - val_loss: 1.4853 - val_accuracy: 0.4788
Epoch 6/9
285/285 [=====] - 101s 355ms/step - loss: 1.5816 - accuracy: 0.3193 - val_loss: 1.5052 - val_accuracy: 0.4788
Epoch 7/9
285/285 [=====] - 101s 354ms/step - loss: 1.5815 - accuracy: 0.3193 - val_loss: 1.5181 - val_accuracy: 0.4788
Epoch 8/9
285/285 [=====] - 100s 351ms/step - loss: 1.5810 - accuracy: 0.3193 - val_loss: 1.4903 - val_accuracy: 0.4788
Epoch 9/9
285/285 [=====] - 99s 348ms/step - loss: 1.5812 - accuracy: 0.3193 - val_loss: 1.5169 - val_accuracy: 0.4788
```

Save 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.

SAVE THE MODEL

```
model.save("arrhythmia.h5")
```

Test The Model:

Load necessary libraries and load the saved model using `load_model`. Taking an image as input and checking the results.

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

TEST THE MODEL

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

```
img=image.load_img("/content/drive/My Drive/Untitled folder/data/test/Left Bundle Branch Block/fig_7769.png",target_size=(64,64))
```

```
img
```

```
x=image.img_to_array(img)
```

```
x
```

```
array([[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]],

      [[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]],

      [[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]],

      ...,

      [[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]],

      [[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]],

      [[255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.],
       ...,
       [255., 255., 255.],
       [255., 255., 255.],
       [255., 255., 255.]]], dtype=float32)
```

```
In [52]: x=np.expand_dims(x,axis=0)
```

```
In [53]: x.ndim
```

```
Out[53]: 4
```

```
In [54]: pred=model.predict(x)
```

```
1/1 [=====] - 0s 257ms/step
```

```
In [55]: pred
```

```
Out[55]: array([[0.05127115, 0.32189828, 0.23932682, 0.1262637 , 0.22656941,
                  0.03467062]], dtype=float32)
```