

FIT3162

User and Technical Guide

Ensemble Deep Convolutional Neural Network for Cardiac
Abnormality Detection using Synchronized Electrocardiogram and
Phonocardiogram Signals

Bryan Ho Yung Kynn

30221455

bhoo0005@student.monash.edu

Chew Shen Min

29640938

sche0094@student.monash.edu

Ryan Chow Shiu Wei

29914124

rcho0018@student.monash.edu

Team: MA_B_7

Supervisor: Dr Ting Chee Ming

GROUP ASSIGNMENT COVER SHEET

Student ID Number	Surname	Given Names
30221455	Ho	Bryan Yung Kynn
29640938	Chew	Shen Min
29914124	Chow	Ryan Shiu Wei

* Please include the names of all other group members.

Unit name and code	FIT3162 – Computer Science Project 2	
Title of assignment	User and Technical Guide	
Lecturer/tutor	Dr Ting Chee Ming	
Tutorial day and time	Wednesday 11am – 12pm	Campus Monash University Malaysia
Is this an authorised group assignment?	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No	
Has any part of this assignment been previously submitted as part of another unit/course?	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No	
Due Date 28-5-2021	Date submitted 28-5-2021	

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date) **Signature of lecturer/tutor**

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity [Policy](#) and [Procedures](#).
 - I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations <http://adm.monash.edu/legal/legislation/statutes>
 - I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
 - No part of this assignment has been previously submitted as part of another unit/course.
 - I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
 - provide to another member of faculty and any external marker; and/or
 - submit it to a text matching software; and/or
 - submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
 - I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.
- Signature Bryan Ho, Shen Min, Ryan Chow Date 28-5-2021
- * delete (iii) if not applicable

Signature _____ Bryan Ho _____ Date: _____ 28-5-2021 _____

Signature _____ Shen Min _____ Date: _____ 28-5-2021 _____

Signature _____ Ryan Chow _____ Date: _____ 28-5-2021 _____

Privacy Statement

The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

Updated: 17 Jun 2014

Table of Contents

1. End User Guide	4
1.1. Convolutional Neural Networks (CNN)	4
1.1.1. Data Preprocessing and Visualization	5
1.1.2. CNN Model Information	7
1.1.3. Training CNN Model	8
1.1.4. Evaluation	9
1.2 Long Short-Term Memory (LSTM)	10
1.3 Graphical Visualizer	11
2. Technical Guide	12
2.1. Hardware Specifications	12
2.2. Software Specifications	12
2.3. Setup and Installation	13
2.3.1. PCG-ECG Dataset	13
2.3.2. Python Installation	13
2.3.3. Setup Python on Command Prompt	15
2.3.4. Setup Virtual Environment	17
2.3.5. Install Necessary Packages	19

1. End User Guide

This software runs on the command-line terminal that performs deep learning binary classification on time-series PCG-ECG data. The software provides 2 different methods of deep learning classification: Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM).

The project has provided a “**training-a**” folder of PCG-ECG data alongside its respective SQI index sheet “**REFERENCE_withSQI.csv**” to allow users to use the software without the need to provide their own dataset. To ensure that the program runs correctly on the command prompt, follow the procedures written in the [Technical Guide](#) first.

1.1. Convolutional Neural Networks (CNN)

To run a CNN deep learning classification on a given PCG-ECG dataset, the following command structure should be typed into the command-line:

```
python cnn.py [Dataset folder name] [SQI CSV file name] [pcg/ecg]
```

Interpretation:

python	The first word that must be written so that the terminal knows that the Python interpreter is used to run this program.
cnn.py	The Python file name to run CNN.
[Dataset folder name]	The folder name of the PCG-ECG dataset. We have provided one named “ training-a ”.
[SQI CSV file name]	The signal quality index (SQI) CSV file name of the PCG-ECG dataset.
[pcg/ecg]	The type of time-series data to be classified. Type “ pcg ” to classify PCG signals, and “ecg” for ECG signals.

For example, to run CNN with the given “**training-a**” folder and “**REFERENCE_withSQI.csv**” file:
For PCG:

```
python cnn.py training-a REFERENCE_withSQI.csv pcg
```

For ECG:

```
python cnn.py training-a REFERENCE_withSQI.csv ecg
```

1.1.1. Data Preprocessing and Visualization

When the program is first run, the command terminal displays the following information after preprocessing:

- Total number of PCG-ECG subjects fed in
- Total number of subjects used and discarded
- Total number of subjects with normal signals
- Total number of subjects with abnormal signals
- The subject with the shortest signal length
- The rounded down value of the shortest signal length

The visualization is different depending on the type of signal being classified due to the different methods of feature extraction made on PCG and ECG respectively.

If PCG is classified, Mel-frequency cepstral coefficients (MFCC) feature extraction is performed and a spectrogram of its extracted features are visualized.

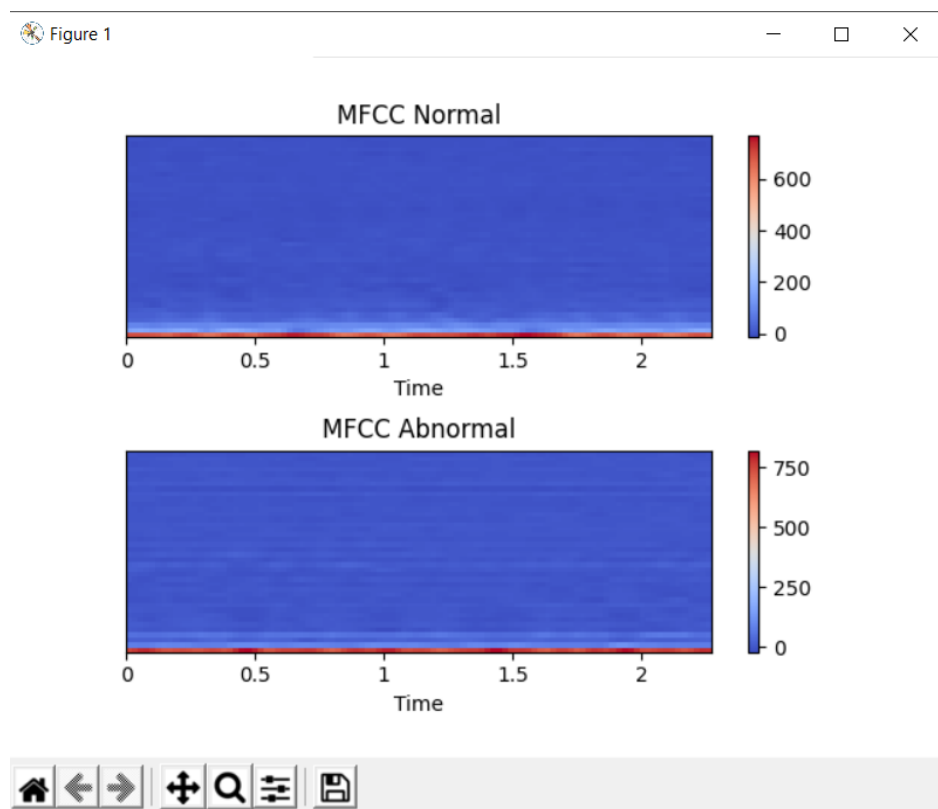


Figure 1.1: Spectrogram of MFCC feature extraction on PCG signals.

If ECG is classified, Wavelet Transform feature extraction is performed and an image on a 2D regular raster is visualized.

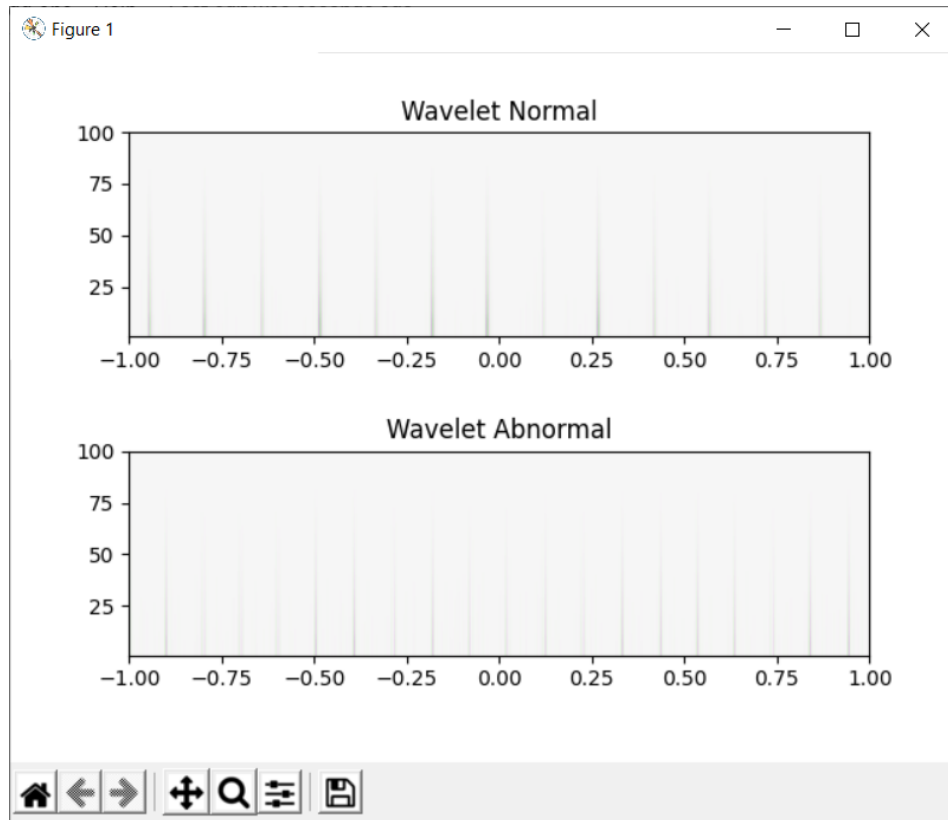


Figure 1.2: Image of Wavelet Transform feature extraction on ECG signals.

The visualizer has additional functions on changing the display of the subplots. For more information on the functions of the visualizer, refer to [1.3 Graphical Visualizer](#).

1.1.2. CNN Model Information

After visualization, a model summary on the CNN model built will be generated:

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 40, 98)]	0
conv1d (Conv1D)	(None, 40, 64)	18880
batch_normalization (Batch Normalization)	(None, 40, 64)	256
re_lu (ReLU)	(None, 40, 64)	0
conv1d_1 (Conv1D)	(None, 40, 64)	12352
batch_normalization_1 (Batch Normalization)	(None, 40, 64)	256
re_lu_1 (ReLU)	(None, 40, 64)	0
conv1d_2 (Conv1D)	(None, 40, 64)	12352
batch_normalization_2 (Batch Normalization)	(None, 40, 64)	256
re_lu_2 (ReLU)	(None, 40, 64)	0
global_average_pooling1d (Global Average Pooling)	(None, 64)	0
dense (Dense)	(None, 2)	130

```
=====  
Total params: 44,482  
Trainable params: 44,098  
Non-trainable params: 384  
=====  
Model summary: 1.1.2. CNN Model Information
```

Figure 1.3: CNN model summary.

The information displayed indicates the type of layers built with their corresponding output shape to be fed to the next layer. It consists of mainly 3 parts:

1. Input layer
2. Middle layer
 - Convolutional layer
 - Batch normalization layer
 - ReLU layer
3. Output layer
 - Global average pooling layer
 - Dense layer

1.1.3. Training CNN Model

The program will then train the CNN model by running 200 epochs while taking into account the training and validation accuracies and losses. If the model starts to overfit, the program will stop automatically.

```
Training Model...
2021-05-13 23:47:11.999940: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Epoch 1/200
9/9 [=====] - 1s 66ms/step - loss: 0.7923 - sparse_categorical_accuracy: 0.3346 - val_loss: 0.8492 - val_sparse_categorical_accuracy: 0.2462
Epoch 2/200
9/9 [=====] - 0s 15ms/step - loss: 0.6074 - sparse_categorical_accuracy: 0.7133 - val_loss: 0.8490 - val_sparse_categorical_accuracy: 0.2462
Epoch 3/200
9/9 [=====] - 0s 14ms/step - loss: 0.6020 - sparse_categorical_accuracy: 0.6784 - val_loss: 0.8760 - val_sparse_categorical_accuracy: 0.2462
Epoch 4/200
9/9 [=====] - 0s 15ms/step - loss: 0.5694 - sparse_categorical_accuracy: 0.6995 - val_loss: 0.8538 - val_sparse_categorical_accuracy: 0.2462
Epoch 5/200
9/9 [=====] - 0s 15ms/step - loss: 0.5457 - sparse_categorical_accuracy: 0.7079 - val_loss: 0.9937 - val_sparse_categorical_accuracy: 0.2462
Epoch 6/200
9/9 [=====] - 0s 15ms/step - loss: 0.5596 - sparse_categorical_accuracy: 0.7057 - val_loss: 0.9047 - val_sparse_categorical_accuracy: 0.2462
Epoch 7/200
9/9 [=====] - 0s 15ms/step - loss: 0.5157 - sparse_categorical_accuracy: 0.7583 - val_loss: 0.9095 - val_sparse_categorical_accuracy: 0.2462
Epoch 8/200
9/9 [=====] - 0s 14ms/step - loss: 0.5194 - sparse_categorical_accuracy: 0.7434 - val_loss: 0.9116 - val_sparse_categorical_accuracy: 0.2462
Epoch 9/200
9/9 [=====] - 0s 15ms/step - loss: 0.5378 - sparse_categorical_accuracy: 0.7313 - val_loss: 0.7523 - val_sparse_categorical_accuracy: 0.2769
Epoch 10/200
9/9 [=====] - 0s 14ms/step - loss: 0.5239 - sparse_categorical_accuracy: 0.7250 - val_loss: 0.7986 - val_sparse_categorical_accuracy: 0.2462
Epoch 11/200
9/9 [=====] - 0s 14ms/step - loss: 0.5021 - sparse_categorical_accuracy: 0.7658 - val_loss: 0.8919 - val_sparse_categorical_accuracy: 0.2462
Epoch 12/200
9/9 [=====] - 0s 14ms/step - loss: 0.4832 - sparse_categorical_accuracy: 0.7475 - val_loss: 0.7963 - val_sparse_categorical_accuracy: 0.2462
Epoch 13/200
9/9 [=====] - 0s 15ms/step - loss: 0.5035 - sparse_categorical_accuracy: 0.7771 - val_loss: 0.9878 - val_sparse_categorical_accuracy: 0.2462
Epoch 14/200
9/9 [=====] - 0s 15ms/step - loss: 0.4901 - sparse_categorical_accuracy: 0.7552 - val_loss: 1.0852 - val_sparse_categorical_accuracy: 0.2462
Epoch 15/200
9/9 [=====] - 0s 15ms/step - loss: 0.4700 - sparse_categorical_accuracy: 0.7432 - val_loss: 1.0050 - val_sparse_categorical_accuracy: 0.2462
Epoch 16/200
9/9 [=====] - 0s 15ms/step - loss: 0.4651 - sparse_categorical_accuracy: 0.7631 - val_loss: 0.9087 - val_sparse_categorical_accuracy: 0.2462
Epoch 17/200
9/9 [=====] - 0s 15ms/step - loss: 0.4525 - sparse_categorical_accuracy: 0.8210 - val_loss: 0.8772 - val_sparse_categorical_accuracy: 0.2615
Epoch 18/200
9/9 [=====] - 0s 14ms/step - loss: 0.4471 - sparse_categorical_accuracy: 0.8170 - val_loss: 0.7985 - val_sparse_categorical_accuracy: 0.2769
Epoch 19/200
9/9 [=====] - 0s 14ms/step - loss: 0.4634 - sparse_categorical_accuracy: 0.7654 - val_loss: 0.7142 - val_sparse_categorical_accuracy: 0.4308
Epoch 20/200
9/9 [=====] - 0s 14ms/step - loss: 0.4217 - sparse_categorical_accuracy: 0.8106 - val_loss: 0.6834 - val_sparse_categorical_accuracy: 0.5231
Epoch 21/200
9/9 [=====] - 0s 15ms/step - loss: 0.4195 - sparse_categorical_accuracy: 0.8056 - val_loss: 1.5028 - val_sparse_categorical_accuracy: 0.2462
Epoch 22/200
9/9 [=====] - 0s 15ms/step - loss: 0.4380 - sparse_categorical_accuracy: 0.7966 - val_loss: 2.0588 - val_sparse_categorical_accuracy: 0.2462
Epoch 23/200
9/9 [=====] - 0s 15ms/step - loss: 0.4562 - sparse_categorical_accuracy: 0.7994 - val_loss: 1.5880 - val_sparse_categorical_accuracy: 0.2462
Epoch 24/200
```

Figure 1.4: Training CNN model over 200 epochs.

1.1.4. Evaluation

The CNN model is then evaluated with its test accuracy and test entropy loss displayed.

```
Evaluating Model...  
3/3 [=====] - 0s 2ms/step - loss: 0.4775 - sparse_categorical_accuracy: 0.8272  
Test Accuracy: 0.8271604776382446  
Test Loss: 0.4775427281856537
```

Figure 1.5: Results on test accuracy and test loss.

In addition, a figure of 2 subplots are displayed which represents the following:

- Training Loss VS Validation Loss
- Training Accuracy VS Validation Accuracy

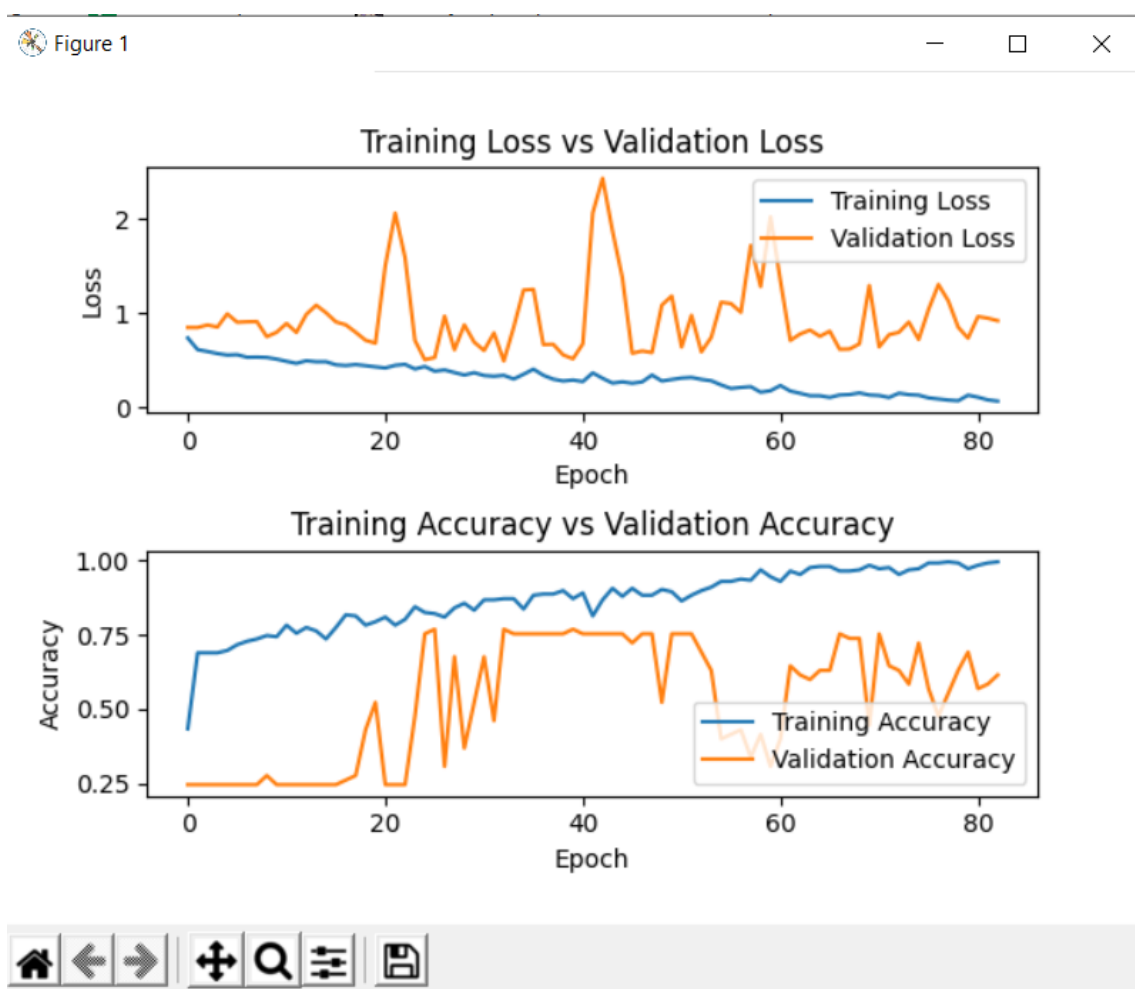


Figure 1.6: Subplots of training-validation loss and accuracy over the course of training the model.

1.2 Long Short-Term Memory (LSTM)

Our LSTM model serves an identical purpose to the CNN model—performing abnormal/normal heartbeat classification on the provided ECG and PCG dataset. Thus the required files and installed software to run the LSTM model are identical to those needed by the CNN model, as described in the first paragraph under section [2.2 Software Specifications](#).

Where the LSTM implementation differs from the CNN implementation is:

- Raw dataset and feature extracted dataset visualization is omitted.
- Training hyperparameters are gathered within a single configuration file—`configs/lstm_config.py`
- Additional performance metrics are displayed to the user during training, namely specificity, sensitivity, F1-score, recall and precision.
- In addition to evaluating the model on the test dataset using `model.evaluate()` we also use it to obtain prediction scores using `model.predict()`

1.2.1 Hyperparameter Configuration File

The goal of this configuration file is to allow the user to conveniently view and configure parameters used for training the model in a single location. This is especially useful when performing model optimization as we did in the `/performance-test` folder.

```
# Training parameters
lstm_config = {
    # Dataset/preprocessing related
    'sample_rate': 1000,
    'min_signal_length': 20000,
    'round_down': False,
    # Feature extraction related
    'n_mfcc': 12,
    'hop_length': 128,
    'widths': 10,
    # Model related
    'optimizer': Adam,
    'learning_rate': 0.0001,
    'best_model_name': 'lstm_best_PCG.h5',
    'mc_monitor': 'val_accuracy',
    'save_weights_only': True,
    'save_best_only': True,
    'mc_verbose': 1,
    'es_monitor': 'val_loss',
    'es_mode': 'auto',
    'loss': 'binary_crossentropy',
    'metrics': ['accuracy', specificity, sensitivity, f1, recall, precision],
    'epochs': 15,
    'batch_size': 16,
    'shuffle': True
}
```

Figure 1.7: Preview of the LSTM hyperparameter configuration file, from `project/configs/lstm_config.py`

1.2.2 LSTM Model Information

Similarly to the CNN model, a summary of the LSTM network architecture is displayed to the user.

```
Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
lstm (LSTM)                  (None, 36, 128)          166400
-----
lstm_1 (LSTM)                (None, 36, 128)          131584
-----
lstm_2 (LSTM)                (None, 256)              394240
-----
dropout (Dropout)           (None, 256)              0
-----
dense (Dense)                (None, 32)               8224
-----
dense_1 (Dense)              (None, 1)                33
-----
Total params: 700,481
Trainable params: 700,481
Non-trainable params: 0
-----
```

Figure 1.7: Preview of LSTM model summary from calling `model.summary()` in `main-lstm.py`

1.2.3 Training LSTM Model

Next, the compiled LSTM model is trained according to the parameters specified in the configuration file (epochs, learning rate, batchsize, etc.).

```
Epoch 1/15
16/16 - 11s - loss: 0.6942 - accuracy: 0.4917 - specificity: 0.5727 - sensitivity: 0.4548 - f1: 0.5523 - recall: 0.4548 - precision: 0.7278 - val_loss: 0.69
21 - val_accuracy: 0.5185 - val_specificity: 0.4500 - val_sensitivity: 0.4301 - val_f1: 0.5049 - val_recall: 0.4301 - val_precision: 0.6204

Epoch 00001: val_accuracy improved from -inf to 0.51852, saving model to lstm_best_PCG.h5
Epoch 2/15
16/16 - 3s - loss: 0.6872 - accuracy: 0.5868 - specificity: 0.5653 - sensitivity: 0.6325 - f1: 0.6859 - recall: 0.6325 - precision: 0.7751 - val_loss: 0.688
3 - val_accuracy: 0.5309 - val_specificity: 0.2111 - val_sensitivity: 0.5180 - val_f1: 0.5503 - val_recall: 0.5180 - val_precision: 0.5892

Epoch 00002: val_accuracy improved from 0.51852 to 0.53086, saving model to lstm_best_PCG.h5
Epoch 3/15
16/16 - 3s - loss: 0.6832 - accuracy: 0.6322 - specificity: 0.4828 - sensitivity: 0.6663 - f1: 0.7149 - recall: 0.6663 - precision: 0.7987 - val_loss: 0.684
5 - val_accuracy: 0.5185 - val_specificity: 0.1500 - val_sensitivity: 0.5316 - val_f1: 0.5487 - val_recall: 0.5316 - val_precision: 0.5707

Epoch 00003: val_accuracy did not improve from 0.53086
Epoch 4/15
16/16 - 3s - loss: 0.6750 - accuracy: 0.7107 - specificity: 0.5106 - sensitivity: 0.7550 - f1: 0.7764 - recall: 0.7550 - precision: 0.8204 - val_loss: 0.676
6 - val_accuracy: 0.6049 - val_specificity: 0.1500 - val_sensitivity: 0.6330 - val_f1: 0.6154 - val_recall: 0.6330 - val_precision: 0.6009
```

Figure 1.8: Preview of training epochs 1-3 out of 15, we also observe the operation of the `ModelCheckpoint` callback which saves the model weights if the model's validation accuracy improves after an epoch.

1.2.4 Post-training LSTM Evaluation

After training has concluded, either due to reaching the number of epochs specified in the configuration file, or when validation loss doesn't improve for the number of epochs specified in the patience parameter of the `EarlyStopping` callback, two plots are shown to the user—first is training accuracy and validation accuracy against epochs, second is training loss and validation loss against epochs.

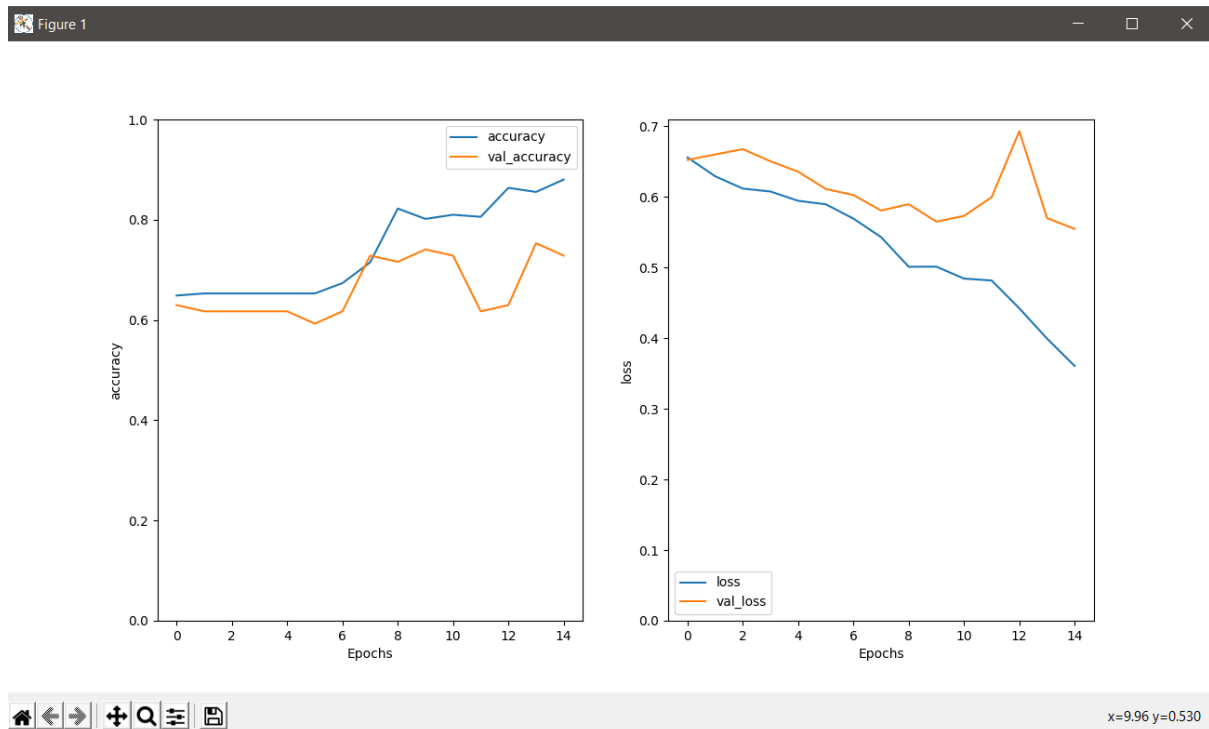


Figure 1.9: Post-training plots for evaluating the model's training, training accuracy and validation accuracy plot on the left and training loss and validation loss plot on the right.

Then, some additional information about the model's training is printed to the terminal as well, as shown in Figure 1.10 below.

```
best epoch was number 8
best epoch had train_acc, loss, spec, sens 0.7066 0.5354 0.7045 0.6721
best epoch had val_acc, loss, spec, sens 0.7531 0.584 0.3639 0.7633
completed epochs 15
```

Figure 1.10: Additional training information displayed after the plots, the epoch where the best model was saved, it is training and validation metrics, and lastly the total completed epochs.

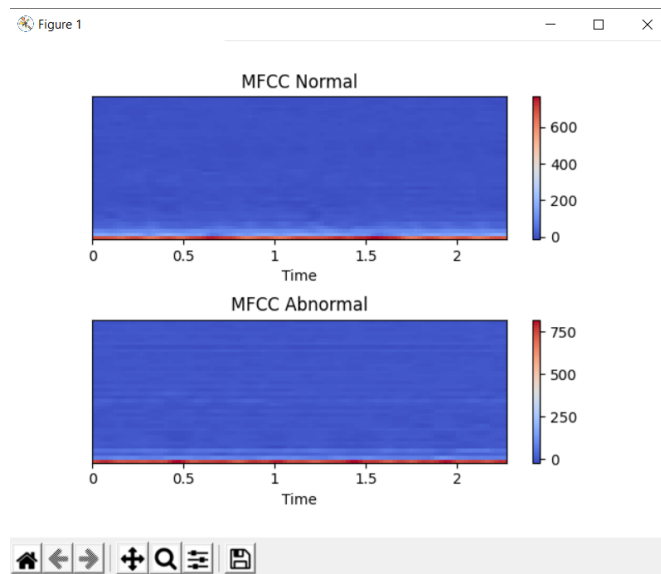
Lastly, the saved model from the best epoch is loaded, then it's performance on the test dataset is obtained using `model.evaluate()` and it's prediction accuracy on the training dataset and the test dataset is computed using `model.predict()`.







```
Saved model performance on test set:
Accuracy: 0.6790123581886292
Loss: 0.6164900660514832
Sensitivity: 0.809855043888092
Specificity: 0.3492063581943512
prediction accuracy on training set: 0.843
prediction accuracy on test set: 0.679 (same as model.evaluate() accuracy score)
```

Figure 1.11: Performance metrics of the saved best performing model on the test dataset, and it is prediction accuracy on the training and test datasets.

1.3 Graphical Visualizer

The visualizer has several buttons at the bottom left corner to modify the display of the plots. Below is a table explaining what each button does.



	Reset the plot to its original view.
	Move the view back and forth.
	<p>Allows for repositioning of the subplot.</p> <ul style="list-style-type: none"> • Hold down the left click and move to pan the view. • Hold down the right click and move to zoom. • Hold down the x key to fix the x-axis. • Hold down the y key to fix the y-axis. • Hold down the Ctrl key to fix the plot's aspect. <p>Note that you can combine the mouse and keyboard buttons for more maneuverability (e.g., x key + left click fixes the x-axis and lets you pan vertically)</p>
	<p>Draw a rectangle to zoom to that area.</p> <ul style="list-style-type: none"> • Hold down the x key to fix the x-axis. • Hold down the y key to fix the y-axis. • Hold down the Ctrl key to fix the plot's aspect.
	Allows configuration on the subplot's dimensions and padding.
	Save the figure as PNG.

2. Technical Guide

2.1. Hardware Specifications

Type	Minimum Requirements
Processor	Intel Core i5 or equivalent
GPU	NVIDIA GeForce GT1030
RAM	8 GB
Storage	10 GB of available space

2.2. Software Specifications

Type	Software
Operating System	Windows 10 Home and above (64-bit)
Programming Language	Python 3.7.1
Software Libraries	<ul style="list-style-type: none">• TensorFlow 2.4.1• tqdm 4.49.0• SciPy 1.5.2• pandas 1.1.2• NumPy 1.19.5• librosa 0.8.0• matplotlib 3.3.2• scikit-learn 0.24.1• Keras 2.4.3

2.3. Setup and Installation

2.3.1. PCG-ECG Dataset

- Ensure that you have the “**training-a**” folder and within it consisting of MATLAB data which corresponds to the patients’ PCG-ECG time frequency data.
- Ensure that you have the “**REFERENCE_withSQL.csv**” CSV file. This consists of the signal quality indices (SQI) referencing to every patient in “**training-a**” respectively.
- DO NOT attempt to modify these folders and files.

2.3.2. Python Installation

1. Click [here](#) to download Python 3.7.1.
2. Scroll down under Files, and click on [Windows x86-64 executable installer](#) to download the Python installer.

Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		99f78ecbfc766ea449c4d9e7eda19e83	22802018	SIG
XZ compressed source tarball	Source release		0a57e9022c07fad3dadb2eef58568edb	16960060	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	ac6630338b53b9e5b9dbb1bc2390a21e	34360623	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	b69d52f22e73e1fe37322337eb199a53	27725111	SIG
Windows help file	Windows		b5ca69aa44aa46cdb8cf2b527d699740	8534435	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	74f919be8add2749e73d2d91eb6d1da5	6879900	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	4c9fd65b437ad393532e57f15ce832bc	26260496	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	6d866305db7e3d523ae0eb252ebd9407	1333960	SIG
Windows x86 embeddable zip file	Windows		aa4188ea480a64a3ea87e72e09f4c097	6377805	SIG
Windows x86 executable installer	Windows		da24541f28e4cc133c53f0638459993c	25537464	SIG
Windows x86 web-based installer	Windows		20b163041935862876433708819c97db	1297224	SIG

Figure 2.1: Files section in the Python 3.7.1 installation website.

3. Run the installer. Check “**Add Python 3.7 to PATH**” and click “**Install Now**”. Be sure to remember the address where your Python is installed as a contingency.

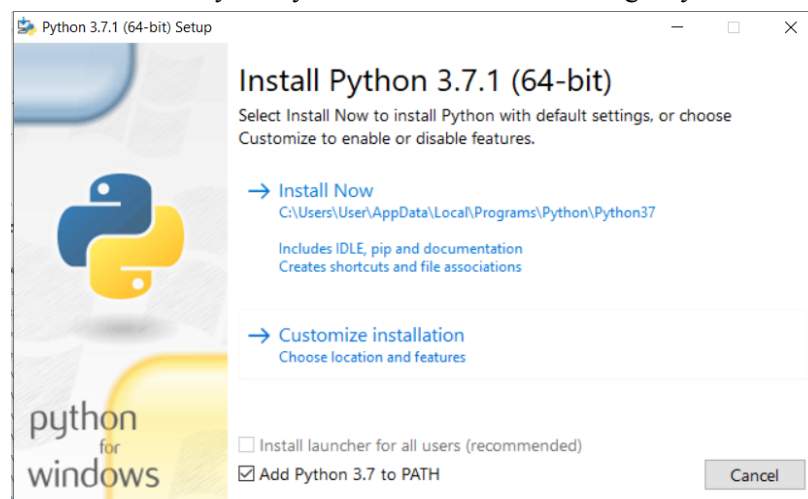


Figure 2.2: Python 3.7.1 installer.

4. Go to the Start search bar and type “**cmd**” to open up the command prompt.

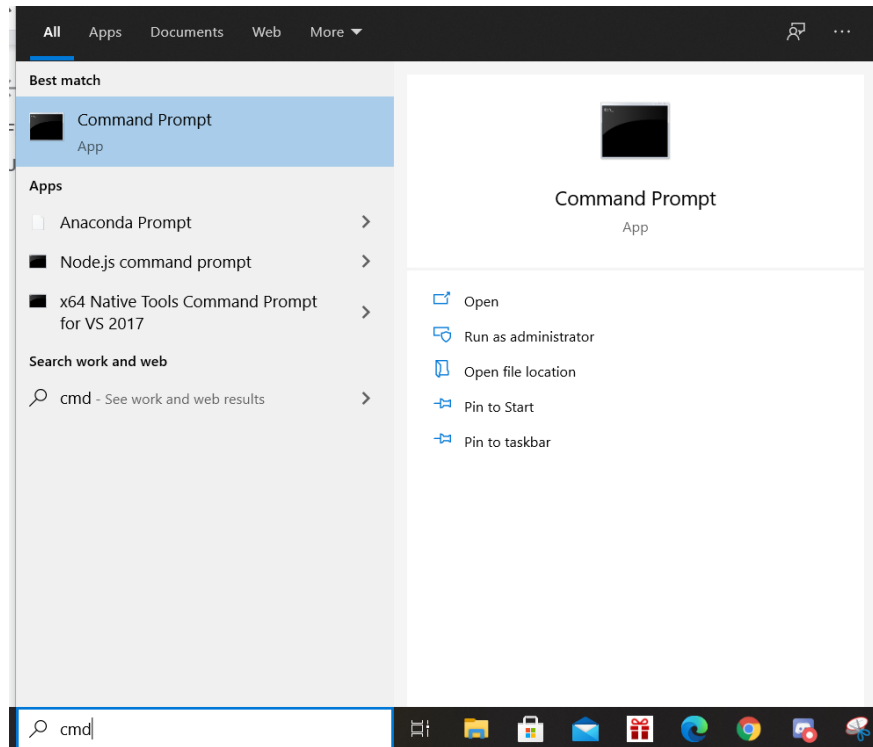


Figure 2.3: “cmd” in Start search bar.

5. Type “**python**” in the terminal. If “**Python 3.7.1**” is displayed, that means you have successfully installed Python to be used on Windows command prompt and you can proceed to [2.3.4. Setup Virtual Environment](#). If you **DO NOT** see a similar statement in Figure 2.4 (or Microsoft Store opens instead), please proceed to [2.3.3. Setup Python on Command Prompt](#).

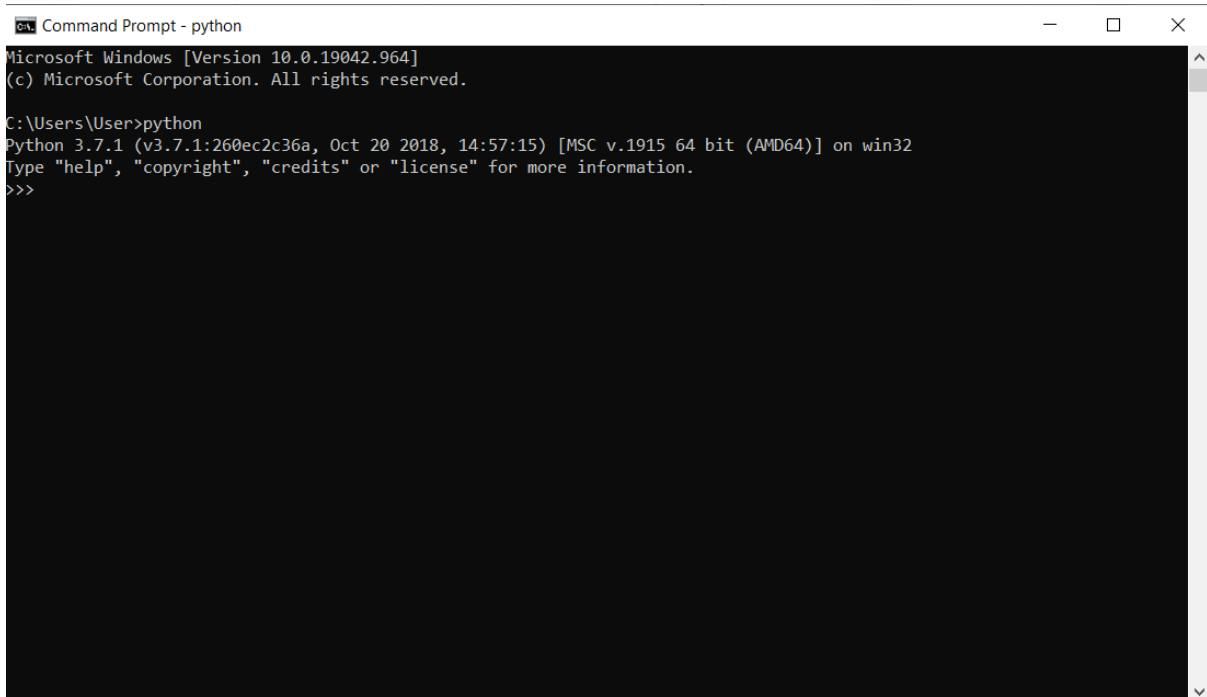


Figure 2.4: Python installed and can be used on Command Prompt.

2.3.3. Setup Python on Command Prompt

1. Head to the location where Python 3.7.1 is installed (the address shown under Install Now in Figure 2.2). Copy the address as text.

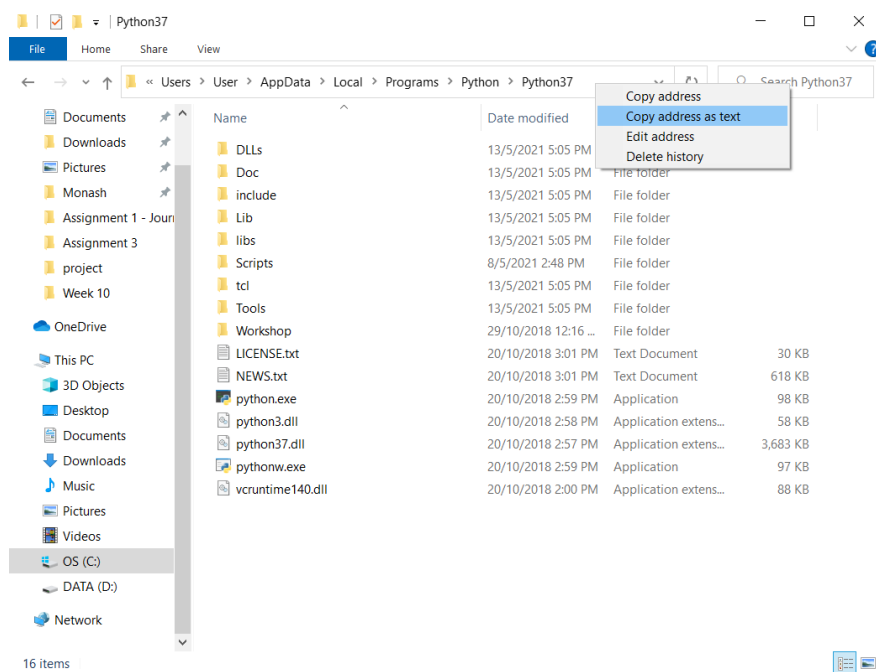


Figure2.5: Copy address as text.

2. Open the Control Panel.
3. Click on “System and Security”.
4. Click on “System”. The About window should pop up. Scroll down and click “Advanced System Settings” under the Related settings section. The “System Properties” window should pop up.

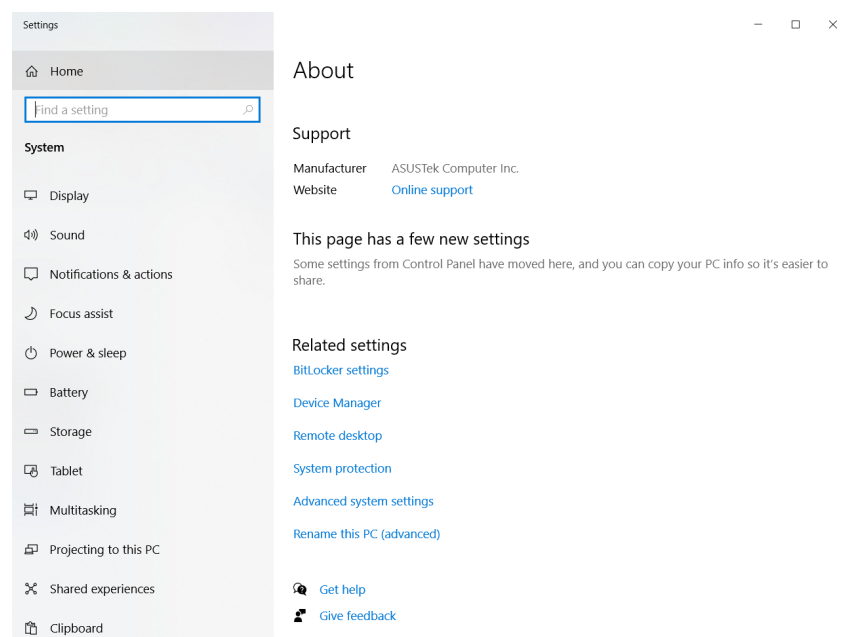


Figure 2.6: Advanced system settings under Related settings.

5. Click on “**Environment Variables...**”. A pop-up window should appear.

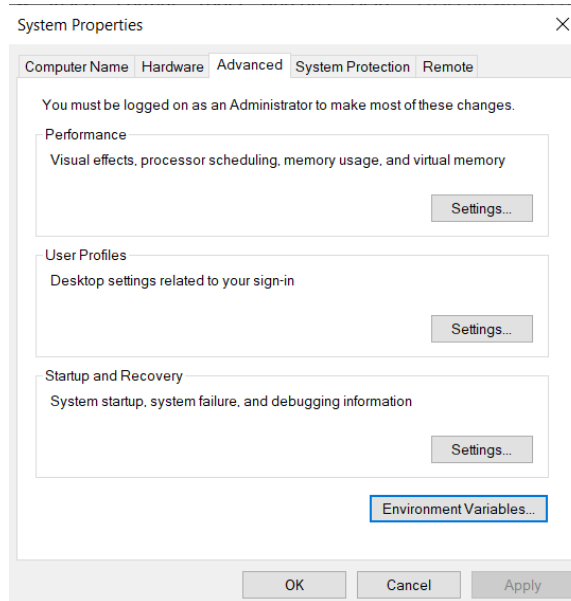


Figure 2.7: Click on *Environment Variables...*

6. Under the “**System variables**” section, double-click on “**Path**”.

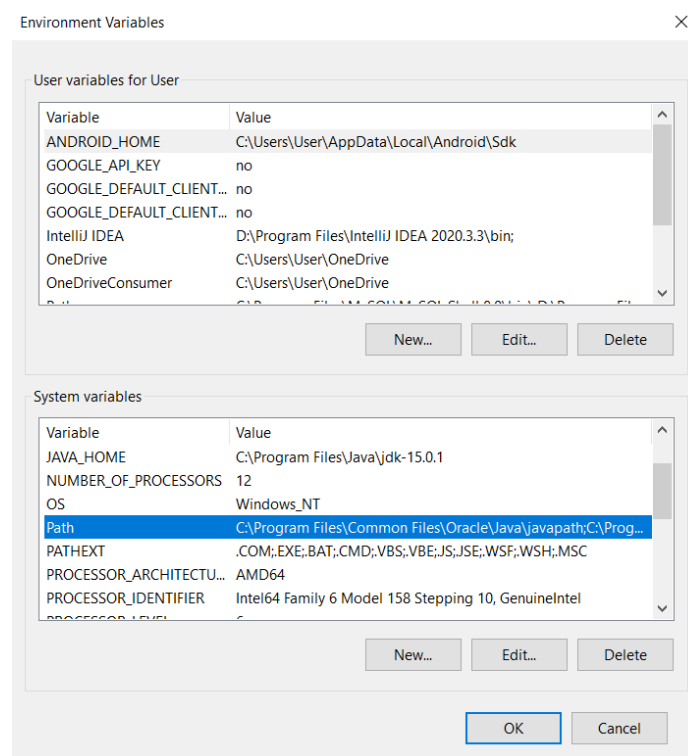


Figure 2.8: Double-click on *Path*.

- Click on “New” and paste in the address of your Python’s directory you previously copied. After that, click “OK”.

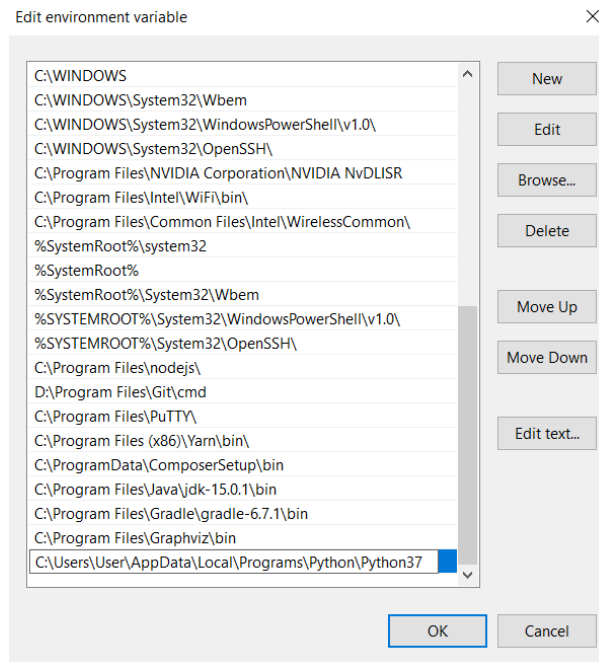


Figure 2.9: Paste address in PATH.

- Click “OK” for all the pop-up windows opened to close them and open the command prompt.
- Type “**python**” in the terminal and you should see “**Python 3.7.1**” displayed. You can now run python scripts on the command terminal.

2.3.4. Setup Virtual Environment

- Open the command prompt in the directory of the project folder.

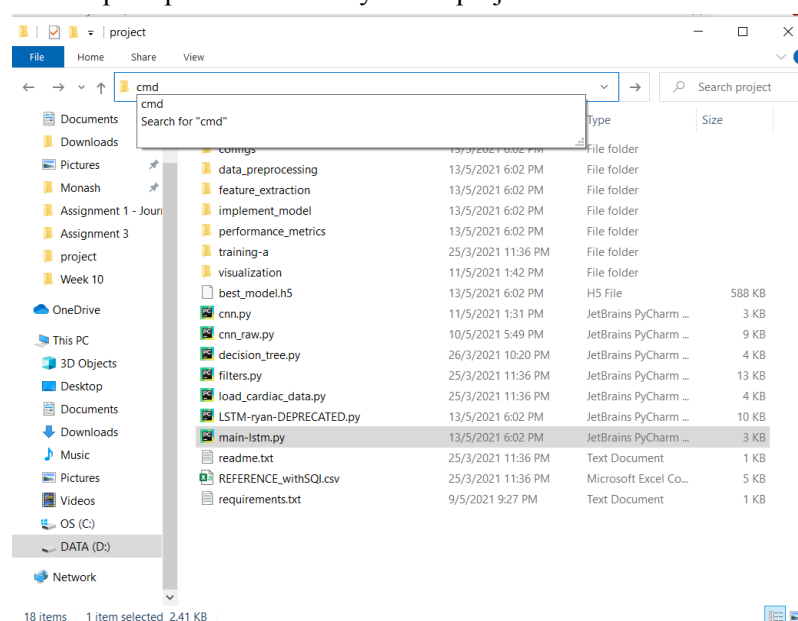


Figure 2.10: Type “cmd” in the project directory above.

2. Type the following line into the command-line:

```
python -m venv env
```

An env folder should be created within the root directory of the project folder.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.964]
(c) Microsoft Corporation. All rights reserved.

D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project>python -m venv env
D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project>
```

Figure 2.11: `python -m venv env`.

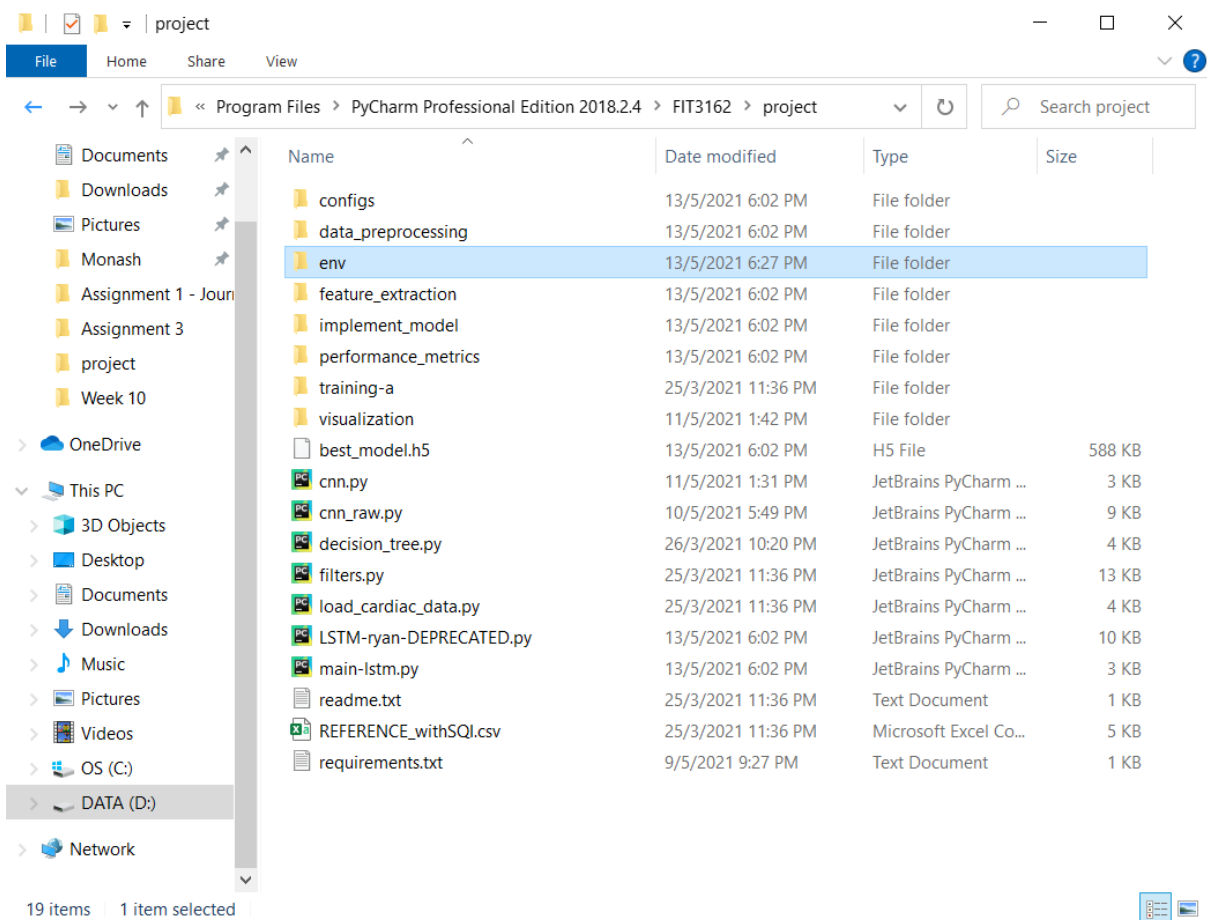


Figure 2.12: `env` folder created.

3. Enter the Scripts folder located in the env folder by typing the following statement in the command-line:

```
cd env/Scripts/
```

```
D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project>cd env/Scripts/
D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project\env\Scripts>
```

Figure 2.13: `cd env/Scripts/`.

4. Type `activate.env` should appear at the front of your command-line.

```
D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project\env\Scripts>activate
(env) D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project\env\Scripts>
```

Figure 2.14: *activate.*

2.3.5. Install Necessary Packages

1. Once you have activated your virtual environment on the command-line, head back to the project folder by typing the following command:

```
cd.. && cd..
```

```
(env) D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project\env\Scripts>cd.. && cd..
(env) D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project>
```

Figure 2.15: *cd.. && cd..*

2. Install the packages listed in “**requirements.txt**” via **pip** by typing the following command:

```
pip install -r requirements.txt
```

```
(env) D:\Program Files\PyCharm Professional Edition 2018.2.4\FIT3162\project>pip install -r requirements.txt
Collecting tensorflow==2.4.1 (from -r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/68/bc/42bbb31a25a708e2e24881724ec7bcea0530492de8b1a2e0d8fe43eb2f6/tensorflow-2.4.1-cp37-win_amd64.whl
Collecting tqdm==4.49.0 (from -r requirements.txt (line 2))
  Using cached https://files.pythonhosted.org/packages/73/d5/f220e0c69b2f346b5649b66abeb391d1a0a59997a7cfc823325bd7a3e/tqdm-4.49.0-py2.py3-none-any.whl
Collecting scipy==1.5.2 (from -r requirements.txt (line 3))
  Using cached https://files.pythonhosted.org/packages/66/80/d8a5050df5b4d8229e018f3222fe603ce7f92c026b78f4e05d69c3a6c43b/scipy-1.5.2-cp37-cp37m-win_amd64.whl
Collecting pandas==1.1.2 (from -r requirements.txt (line 4))
  Using cached https://files.pythonhosted.org/packages/c5/16/07da3435a161ae411eef63d6c5edcf9fd11a8a11e94f60d259693b7e0804/pandas-1.1.2-cp37-cp37m-win_amd64.whl
Collecting numpy==1.19.5 (from -r requirements.txt (line 5))
  Using cached https://files.pythonhosted.org/packages/ff/18/60ac053857fb924b0324c81200b59c00317ebaa3c14b7478266b50ffed19/numpy-1.19.5-cp37-cp37m-win_amd64.whl
Collecting librosa==0.8.0 (from -r requirements.txt (line 6))
  Using cached https://files.pythonhosted.org/packages/26/4d/c22d8ca74ca2c13d4acc430fa353954886104321877b65fa871939e78591/librosa-0.8.0.tar.gz
Collecting matplotlib==3.2 (from -r requirements.txt (line 7))
  Using cached https://files.pythonhosted.org/packages/81/51/dadfa433ec13d9966393b1726a5c22d77343aa7797b4898965f32e07/matplotlib-3.2.2-cp37-cp37m-win_amd64.whl
Collecting grpcio==1.32.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/67/5f/bf8222117f90a2f6d0f8f43bda3b084d7b24b6d5c84dbc6e6c9df4c742/grpcio-1.32.0-cp37-win_amd64.whl
Collecting tensorboard==2.4 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/44/f5/7feea02a3fb54d5db827ac4b822a7ba8933826b36de21880518250b8733a/tensorboard-2.5.0-py3-none-any.whl
Collecting flatbuffers==1.12.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/cb/26/712e578c5f14e26ae314c39a1bdc4eb2ec2f4ddc89b708c78e0a0d20423/flatbuffers-1.12-py2.py3-none-any.whl
Collecting opt-einsum==3.3.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/bc/19/404708a7e54ad2798907210462f4950c3442ea51acc8790f3da48d2bee8b/opt_einsum-3.3.0-py3-none-any.whl
Collecting tensorflow-estimator==2.4.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/74/7c/622d9849abf3efb81e482ff170758742e392ee129ce1540611199a59237/tensorflow_estimator-2.4.0-py2.py3-none-any.whl
Collecting termcolor==1.1.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/8a/48/a70be51647d0eb9f10e2a4511bf3ff8c1eb14e9e4fab46173aa79f981/termcolor-1.1.0.tar.gz
Collecting keras-preprocessing==1.1.2 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/79/4c/7c3275a01e2ef9368a892926ab932b33bb13d55794881e3573482b378a7/Keras_Preprocessing-1.1.2-py2.py3-none-any.whl
Collecting absl-py==0.8 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/92/c9/ef0fae29182a7a807d203f0eff8296b6da92098cc41db33a434f4be84bf/absl_py-0.12.0-py3-none-any.whl
Collecting typing-extensions==3.7.4 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/60/7a/e8815abb54db0e6671ab088d079c57ce54e8a01a3ca443f561ccadb37e/typing_extensions-3.7.4.3-py3-none-any.whl
Collecting six==1.15.0 (from tensorflow==2.4.1->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/ee/ff/48ade5e0813944729fe4b0310ba2a27474b3ff1c52d924a8a4cb04078a/six-1.15.0-py2.py3-none-any.whl
Collecting gast==0.3.3 (from tensorflow==2.4.1->-r requirements.txt (line 1))
```

Figure 2.16: *pip install -r requirements.txt.*

3. Once the packages are installed, the setup is complete and you can start running the program.