NeuroBrave Job Assignment

1.

```python
from pynput.mouse import Listener
import os
import json
import smtplib, ssl
import time

boolfindjson = 1  # if no .json file exist
filesinDirectory = os.listdir()  # Read files in directory
for files in filesinDirectory:  # run on files in directory
    if str(files).find('.json') > 0:  # Check if there is a .json file
        boolfindjson = 0
        with open(files) as f:  # Read .json file
            data = json.load(f)

if boolfindjson:  # If no .json file found
    print("No suitable .json file")
    import sys

    sys.exit()  # stop executing


def on_move(x, y):  # if mouse moves on screen
    print('Pointer moved to {0}'.format((x, y)))
    if float(data['x1']) < x < float(data['x2']) \
            and float(data['y1']) < y < float(data['y2']):  # If mouse is in
rectangle
        sendEmail()
        time.sleep(30)


def sendEmail():  # sends email
    port = 587  # For starttls
    smtp_server = "smtp.gmail.com"
    sender_email = "nircaf2@gmail.com"
    receiver_email = data['target_email']  # get target email from .json file
    password = "nir159Caf"
    message = """\
    Subject: Hi NeuroBrave

    This message is sent from Python."""  # Email message

    context = ssl.create_default_context()
    with smtplib.SMTP(smtp_server, port) as server:
        server.ehlo()  # Can be omitted
        server.starttls(context=context)
        server.ehlo()  # Can be omitted
```

```
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message)
    print("Email sent from {0} to {1}".format(sender_email, receiver_email))


# Collect events until released
with Listener(  # Refreshes mouse move
        on_move=on_move,
) as listener:
    listener.join()
```

2.

The data contains 2670 scans of 15 subjects. Each scan can be shown as a 32X32 pixels picture given the RGB values.
Examples for the images that were given to classify:

Because the data has already been transformed into images I chose to deal with the difference between the images with a convolution neural network.
Four mental situations labeled for subjects were classified using CNN. Both inter and intra subjects' machine learning weights were calculated and compared in order to achieve best fitted results.
I used Tensorflow's Keras library to build the machine learning algorithm

Loading Data:

```
features = scipy.io.loadmat('FeatureMat_timeWin.mat')['features']
img = scipy.io.loadmat('images.mat')['img']
subjNum = scipy.io.loadmat('trials_subNums.mat')['subjectNum'][0]
Label = (features[:, -1] - 1).astype(int)
```

Reshaping data:
testtrain function - gets all the images and labels and splits them to train and test. 70% of the data to train the weights and 30% to verify the accuracy of those weights. the images afterwards are re-shaped and normalized for max value 1 per image.
The function returns training and testing images and labels.

```
def testtrain(img, label):
    train, test = train_test_split(range(0, len(img)), test_size=0.3)
    train_images = img[train]
    train_labels = Label[train]
```

```
    test_images = img[test]
    test_labels = Label[test]

    # Normalize the images.
    train_images = (train_images / numpy.max(train_images))
    test_images = (test_images / numpy.max(test_images))

    # Set input shape
    sample_shape = train_images[0].shape
    img_width, img_height = sample_shape[1], sample_shape[2]
    input_shape = (img_width, img_height, 3)

    # Reshape data
    train_images = train_images.reshape(len(train_images), input_shape[0],
input_shape[1], input_shape[2])
    test_images = test_images.reshape(len(test_images), input_shape[0],
input_shape[1], input_shape[2])
    return train_images, train_labels, test_images, test_labels
```

Learning algorithm:

Buildmodel function - the function builds the structure for the model.

- Filter number is optimally very high but comes with the price of slowing down learning.
- Optimiser Nadam - gradient descent for learning with momentum for direction of image change.
- Loss function - ‘sparse’ reflects numerical labeling
- validation split - in training the model we use part of the data for cross validation. This increases accuracy and reduces computation power.

gets training and testing images and labels

```
def buildModel():
    num_filters = 8  # filters that the convolutional layer will learn.
    kernel_size = 3  # specifying the width and height of the 2D convolution
window.
    pool_size = 1   # reduce the spatial dimensions of the output volume.
    dilation_rate = (5,5)  #  a basic convolution only applied to the input
volume with defined gaps
    # Build the model.
    model = Sequential([
        Conv2D(num_filters,
               kernel_size,
               input_shape=(32, 32, 3),
               strides=(1, 1),  # specifying the "step" of the convolution along
the x and y axis of the input volume.
               padding='same',  # zero-padded
               dilation_rate=dilation_rate,
               ),
        MaxPooling2D(pool_size=(pool_size, pool_size)),  # reduce the spatial
dimensions of the output volume.
```

```
        Flatten(),
        Dense(512, activation='relu'),
        Dense(16, activation='softmax'),
    ])
    model.summary()
    # # Compile the model.
    model.compile(
        'Nadam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'],
    )
    return model
```

2 different ways to compute weights: inter and intra.

Intra - for subjects with a big difference from the group a customized model based on their scans only is required to get to optimal results.
Pros - customizable per subject. Can handle big changes between subjects.
Cons - Demands a lot of computer power and time. Each subject has a small amount of data.

```
def runPerSubj(img, label):
    train_images, train_labels, test_images, test_labels = testtrain(img, label)
    model = buildModel()
    # Train the model.
    model.fit(
        train_images,
        train_labels,
        epochs=10,
        validation_split=0.15,
        validation_data=(test_images, test_labels),
    )
    # Predict on the first 5 test images.
    predictions = model.predict(test_images)
    predictionsToCompare = numpy.argmax(predictions, axis=1)
    # Print our model's predictions.
    print(predictionsToCompare[:5])  #
    # Check our predictions against the ground truths.
    print(test_labels[:5])  #
    return numpy.average(test_labels == predictionsToCompare)
```

Inter - overall model from the data of all the subjects. New runs use stored weights and save power.
Pros - more likely to fit new subjects.

```
def loadWeights(img, label):
    train_images, train_labels, test_images, test_labels = testtrain(img, label)
    model = buildModel()
    model.load_weights('cnn.h5')
    # Predict on the first 5 test images.
```

```
    predictions = model.predict(test_images)
    predictionsToCompare = numpy.argmax(predictions, axis=1)
    # Print our model's predictions.
    print(predictionsToCompare[:5])  #
    # Check our predictions against the ground truths.
    print(test_labels[:5])  #
    return numpy.average(test_labels == predictionsToCompare)
```

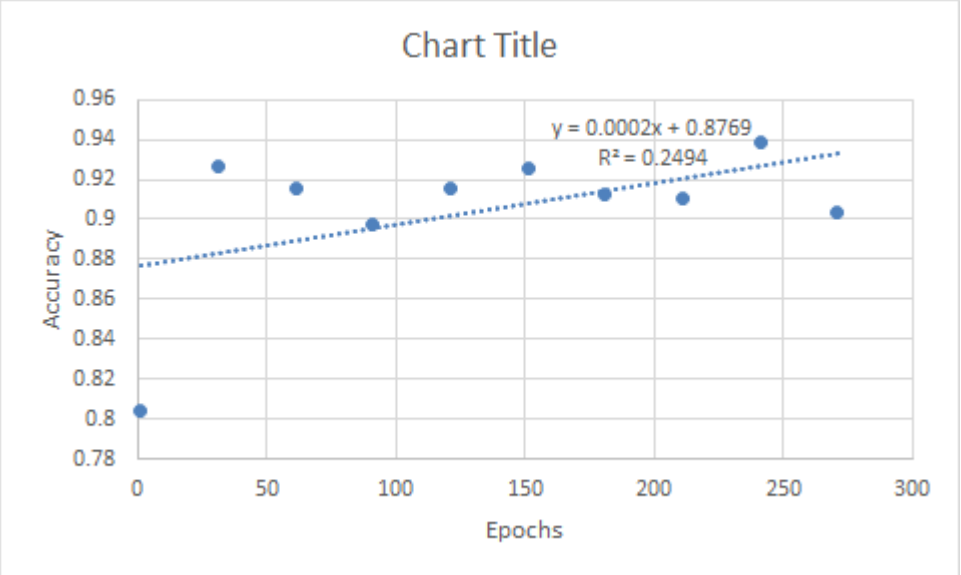
After running the intra model, the algorithm compares between the two weights sets per subject and prints which one is preferable for the subject:

```
else:  # Per Subject
    for subject in range(1, numpy.max(subjNum)):
        labelPos = numpy.where(subjNum == subject)
        if numpy.size(labelPos):
            accuracy = runPerSubj(img[labelPos], Label[labelPos])
            print("Subject number: {0} inter accuracy learning is:
{1}%".format(subject, str(accuracy * 100)))
            accuracyLoadWeights = loadWeights(img[labelPos], Label[labelPos])
            print(
                "Subject number: {0} intra accuracy learning is:
{1}%".format(subject, str(accuracyLoadWeights * 100)))
            if accuracyLoadWeights < accuracy:
                print("Subject number: {0} individual accuracy is better by :
{1}%"
                      .format(subject, str(accuracyLoadWeights - accuracy)))
            else:
                print("Subject number: {0} overall accuracy is better by : {1}%"
                      .format(subject, str(accuracy - accuracyLoadWeights)))
```

Ways to improve learning - most ways require additional computing power.

- More data
- More epochs
- Higher number of filters
- More/bigger dense layers.
- Additional MaxPooling

Inter analysis



Graph of accuracy against epochs

A positive slope shows the correlation between higher number of epochs results in better accuracy.

Overall accuracy average - 90.4%, Std-3.5%

Best accuracy - 93.88% prediction accuracy.

Intra analysis

Example:

Subject output:

Subject number: 14 inter accuracy learning is: 22.22222222222222%

Subject number: 14 individual accuracy is better by : 0.7777777777777778%

Inter output:

inter algorithm preferable for 12 subjects

Algorithms are equal at 3 subjects

Average inter accuracy 1.0

Average intra accuracy 0.35714285714285715

Conclusion - we can see that per subject weights significantly anticipate results more accurately. Unfortunately due to the low number of scans per subject this case shows overfitting and probably would not fit to other subjects.

3. Harmonics were constructed from sin function.

T (time) runs from 0 to 1

X- fundamental function - sin(200πt) for 100Hz peak amplitude

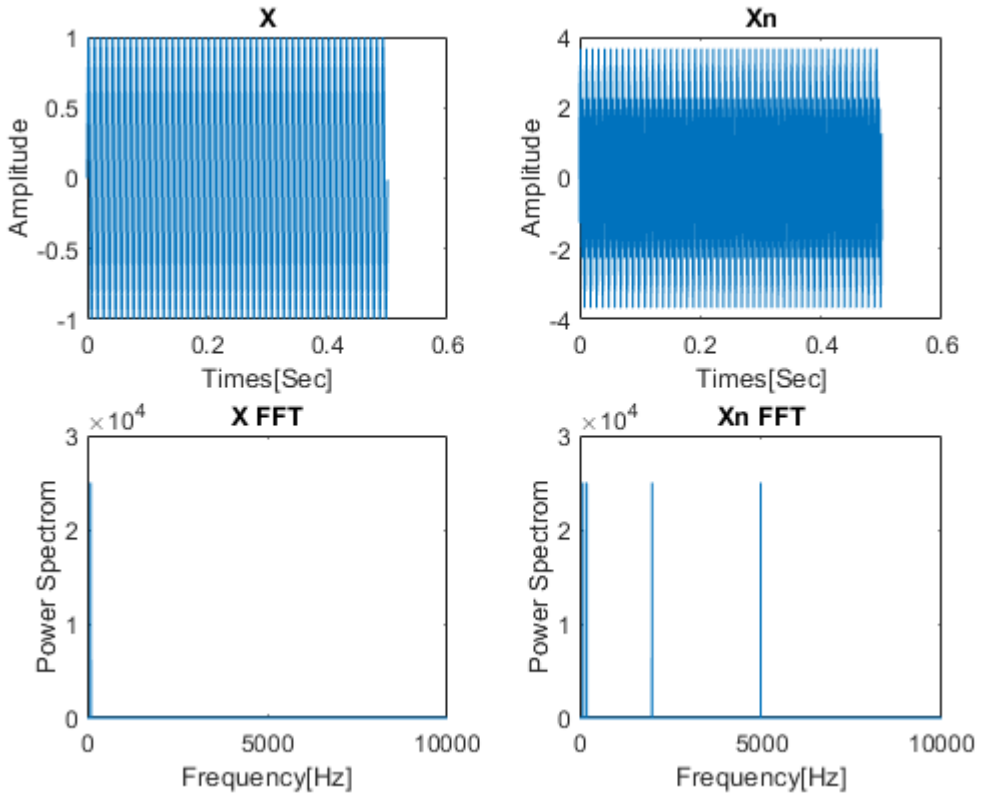
X1 - sin(400πt) for 200 Hz peak amplitude

X2- sin(4000πt) for 2000 Hz peak amplitude

X3- sin(10000πt) for 5000 Hz peak amplitude

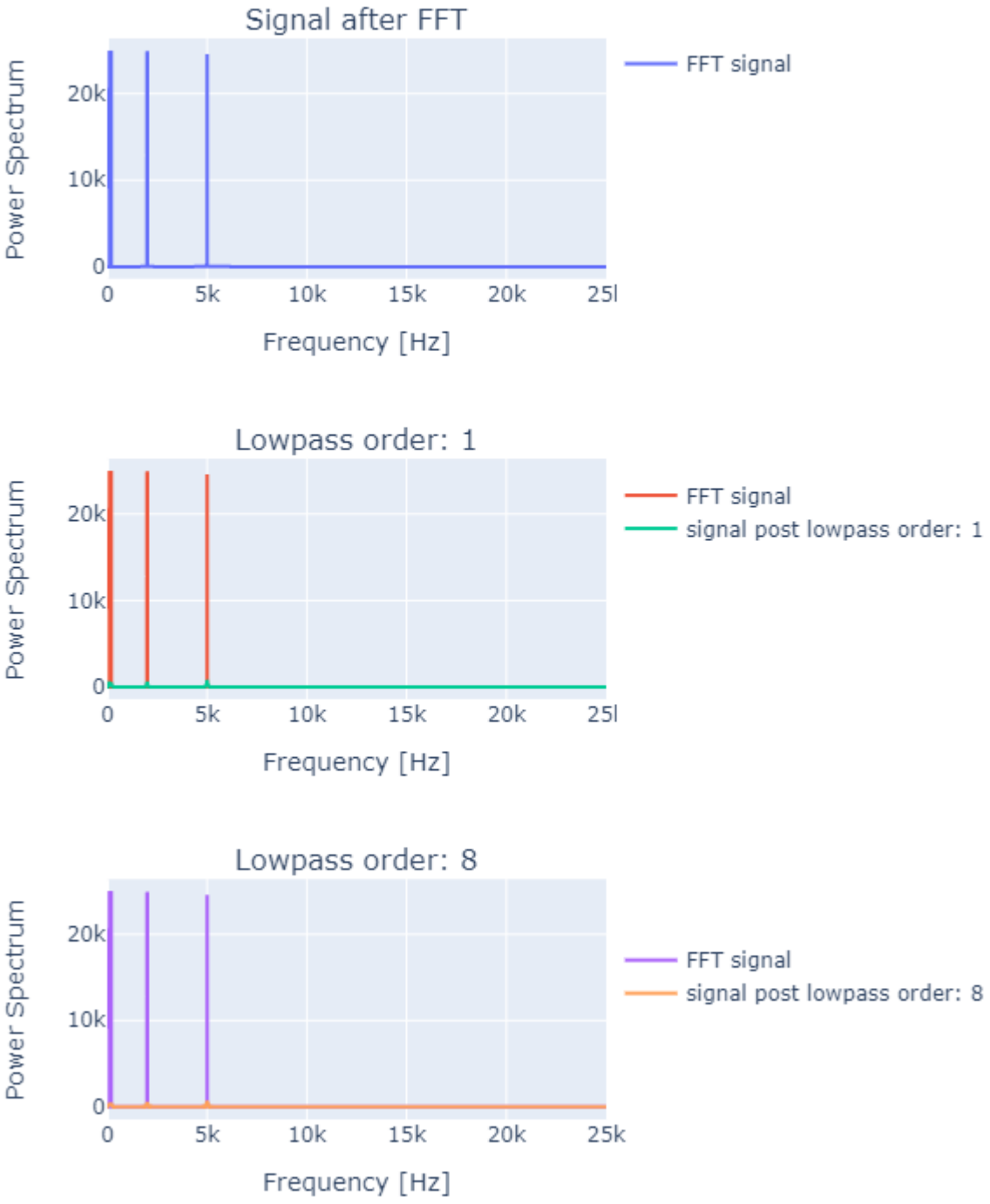
Xn = X1+X2+X3

Since sin is a symmetrical function that it was sliced in half for only the first half of harmonics.



3. 2. Lowpass filter for 300[Hz] plummets power values after 300[Hz]
Spikes are seen at 2[KHz] and 5[KHz] as was seen in Xn FFT.

Harmonics with lowpass



$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

s = Laplace operator

ω_n = natural frequency; $\omega_n = 2\pi f_n$

ζ = damping ratio (called Zeta in the block menu)

Higher lowpass order increases in the negative direction the slope of decreasing the signal. In other words it decreases the slope to be more negative. It adds additional resistance to the circuit past given frequencies.

In case that negative values can exist, low pass after 300[Hz] would look like this: