Description

To ensure the predictions we decided to supplement our EEG Machine Learning models with a CNN model based on the two different classes given by the writing samples.

Importing the Libraries

In [19]:

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

Data Preprocessing

Preprocessing the Training Set

The input image is on RGB. Every image is made up of pixels that range from 0 to 255. We need to normalize them i.e convert the range between 0 to 1 before passing it to the model.

In [20]:

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('Dataset/Training_Set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 146 images belonging to 2 classes.
```

The total of images in the training set is given by 73 * 2 = 146.

Preprocessing the Test Set

In [21]:

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('Dataset/Test_Set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 34 images belonging to 2 classes.
```

The total of images in the test set is given by 16 * 2 = 34.

Building the CNN Model

In [22]:

```
# Initializing the Model
Model = tf.keras.models.Sequential()
```

In [23]:

```
# Adding First Convolution Layer
Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
```

The convolution layer is the layer where the filter is applied to our input image to extract or detect its features. A filter is applied to the image multiple times and creates a feature map which helps in classifying the input image.

In [24]:

```
# Pooling the First Layer
Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

The pooling layer is applied after the Convolutional layer and is used to reduce the dimensions of the feature map which helps in preserving the important information or features of the input image and reduces the computation time.

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.

In [25]:

```
# Adding a Second Convolutional Layer
Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
```

In [26]:

```
# Pooling the Second Layer
Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

In [27]:

```
# Flattening
Model.add(tf.keras.layers.Flatten())
```

The flattening step involves taking the pooled feature map that is generated in the pooling step and transforming it into a one-dimensional vector. This vector will now become the input layer of an artificial neural network.

In [28]:

```
# Full Connection
Model.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

The full connection step involves chaining an artificial neural network onto our existing convolutional neural network.

In [29]:

```
# Output Layer
Model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

We add the output layer. In this project we must have units=1 because we need to classify 2 different classes.

In [30]:

```
# Compiling the CNN
Model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the CNN

In [31]:

```
Model.fit(x = training_set, validation_data = test_set, epochs = 50, verbose= False)
```

Out[31]:

```
<keras.callbacks.History at 0x1bb5951b100>
```

Evaluating the Model

Firstly we evaluate the ability of the model in predicting our training set.

In [32]:

```
acc_training = Model.evaluate(training_set)
print ("The accurancy of the model on the training set is", round(acc_training[1]*100, 2), "%")
```

```
5/5 [==============================] - 4s 681ms/step - loss: 0.5529 - accuracy: 0.7192
The accurancy of the model on the training set is 71.92 %
```

In [33]:

```
acc_test = Model.evaluate(test_set)
print ("The accurancy of the model on the test set is", round(acc_test[1]*100, 2), "%")
```

```
2/2 [==============================] - 1s 51ms/step - loss: 0.5589 - accuracy: 0.6765
The accurancy of the model on the test set is 67.65 %
```