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 [1]:

```python
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 [2]:

```python
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 [3]:

```python
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

Initializing the Model

In [4]:

```python
Model = tf.keras.models.Sequential()
```

Adding First Convolution Layer

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 [5]:

```python
Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
```

Pooling the First Layer

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 [6]:

```python
Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Adding a Second Convolutional Layer

In [7]:

```python
Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
```

Pooling the Second Layer

In [8]:

```python
Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Flattening

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 [9]:

```python
Model.add(tf.keras.layers.Flatten())
```

Full Connection

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

In [10]:

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

Output Layer

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

In [11]:

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

Compiling the CNN

In [12]:

```python
Model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the CNN

In [13]:

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

```
Epoch 1/50
5/5 [==============================] - 7s 1s/step - loss: 0.8415 - accuracy: 0.5137 - val_loss: 0.6979 - val_ac
curacy: 0.5000
Epoch 2/50
5/5 [==============================] - 4s 885ms/step - loss: 0.6966 - accuracy: 0.5000 - val_loss: 0.6958 - val
_accuracy: 0.5000
Epoch 3/50
5/5 [==============================] - 5s 1s/step - loss: 0.6956 - accuracy: 0.5000 - val_loss: 0.6930 - val_ac
curacy: 0.5000
Epoch 4/50
5/5 [==============================] - 4s 987ms/step - loss: 0.6942 - accuracy: 0.4863 - val_loss: 0.6927 - val
_accuracy: 0.5000
Epoch 5/50
5/5 [==============================] - 4s 870ms/step - loss: 0.6938 - accuracy: 0.5000 - val_loss: 0.6929 - val
_accuracy: 0.5000
Epoch 6/50
5/5 [==============================] - 4s 884ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6927 - val
_accuracy: 0.5000
Epoch 7/50
5/5 [==============================] - 5s 914ms/step - loss: 0.6936 - accuracy: 0.4589 - val_loss: 0.6929 - val
_accuracy: 0.5000
Epoch 8/50
5/5 [==============================] - 5s 919ms/step - loss: 0.6935 - accuracy: 0.5000 - val_loss: 0.6934 - val
_accuracy: 0.5000
Epoch 9/50
5/5 [==============================] - 4s 917ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6932 - val
_accuracy: 0.5000
Epoch 10/50
5/5 [==============================] - 5s 921ms/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6928 - val
_accuracy: 0.5000
Epoch 11/50
5/5 [==============================] - 5s 952ms/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6923 - val
_accuracy: 0.5000
Epoch 12/50
5/5 [==============================] - 5s 1s/step - loss: 0.6933 - accuracy: 0.5068 - val_loss: 0.6924 - val_ac
curacy: 0.5000
Epoch 13/50
5/5 [==============================] - 5s 1s/step - loss: 0.6929 - accuracy: 0.5000 - val_loss: 0.6914 - val_ac
curacy: 0.5000
Epoch 14/50
5/5 [==============================] - 5s 949ms/step - loss: 0.6926 - accuracy: 0.5000 - val_loss: 0.6905 - val
_accuracy: 0.6471
Epoch 15/50
5/5 [==============================] - 5s 932ms/step - loss: 0.6925 - accuracy: 0.5205 - val_loss: 0.6902 - val
_accuracy: 0.5000
Epoch 16/50
5/5 [==============================] - 4s 911ms/step - loss: 0.6923 - accuracy: 0.5000 - val_loss: 0.6904 - val
_accuracy: 0.5000
Epoch 17/50
5/5 [==============================] - 5s 970ms/step - loss: 0.6935 - accuracy: 0.4726 - val_loss: 0.6920 - val
_accuracy: 0.6471
Epoch 18/50
5/5 [==============================] - 5s 975ms/step - loss: 0.6927 - accuracy: 0.6233 - val_loss: 0.6921 - val
_accuracy: 0.5294
Epoch 19/50
5/5 [==============================] - 5s 1s/step - loss: 0.6922 - accuracy: 0.5205 - val_loss: 0.6915 - val_ac
curacy: 0.5294
Epoch 20/50
5/5 [==============================] - 5s 929ms/step - loss: 0.6916 - accuracy: 0.5753 - val_loss: 0.6908 - val
_accuracy: 0.5294
Epoch 21/50
5/5 [==============================] - 5s 1s/step - loss: 0.6908 - accuracy: 0.6575 - val_loss: 0.6893 - val_ac
curacy: 0.6176
Epoch 22/50
5/5 [==============================] - 4s 882ms/step - loss: 0.6908 - accuracy: 0.5616 - val_loss: 0.6875 - val
_accuracy: 0.7059
Epoch 23/50
5/5 [==============================] - 4s 906ms/step - loss: 0.6901 - accuracy: 0.5890 - val_loss: 0.6833 - val
_accuracy: 0.5294
Epoch 24/50
5/5 [==============================] - 5s 1s/step - loss: 0.6851 - accuracy: 0.6438 - val_loss: 0.6785 - val_ac
curacy: 0.5588
Epoch 25/50
5/5 [==============================] - 5s 1s/step - loss: 0.6808 - accuracy: 0.5616 - val_loss: 0.6740 - val_ac
curacy: 0.6765
Epoch 26/50
5/5 [==============================] - 4s 967ms/step - loss: 0.6755 - accuracy: 0.6438 - val_loss: 0.6607 - val
_accuracy: 0.7353
Epoch 27/50
5/5 [==============================] - 5s 946ms/step - loss: 0.6721 - accuracy: 0.6507 - val_loss: 0.6471 - val
_accuracy: 0.6471
Epoch 28/50
5/5 [==============================] - 5s 933ms/step - loss: 0.6534 - accuracy: 0.6507 - val_loss: 0.6333 - val
_accuracy: 0.7353
Epoch 29/50
5/5 [==============================] - 5s 1s/step - loss: 0.6557 - accuracy: 0.6849 - val_loss: 0.6384 - val_ac
curacy: 0.6765
Epoch 30/50
5/5 [==============================] - 5s 1s/step - loss: 0.6453 - accuracy: 0.6233 - val_loss: 0.6232 - val_ac
curacy: 0.7059
Epoch 31/50
5/5 [==============================] - 5s 961ms/step - loss: 0.6391 - accuracy: 0.7123 - val_loss: 0.6121 - val
_accuracy: 0.7353
Epoch 32/50
5/5 [==============================] - 5s 958ms/step - loss: 0.6367 - accuracy: 0.6986 - val_loss: 0.6002 - val
_accuracy: 0.7353
Epoch 33/50
5/5 [==============================] - 5s 935ms/step - loss: 0.6224 - accuracy: 0.7055 - val_loss: 0.5930 - val
_accuracy: 0.6471
Epoch 34/50
5/5 [==============================] - 5s 1s/step - loss: 0.6156 - accuracy: 0.6849 - val_loss: 0.6104 - val_ac
curacy: 0.6176
Epoch 35/50
5/5 [==============================] - 5s 980ms/step - loss: 0.6180 - accuracy: 0.6986 - val_loss: 0.5775 - val
_accuracy: 0.7059
Epoch 36/50
5/5 [==============================] - 5s 987ms/step - loss: 0.6191 - accuracy: 0.6712 - val_loss: 0.5788 - val
_accuracy: 0.6765
Epoch 37/50
5/5 [==============================] - 5s 962ms/step - loss: 0.6260 - accuracy: 0.7123 - val_loss: 0.5628 - val
_accuracy: 0.7059
Epoch 38/50
5/5 [==============================] - 4s 908ms/step - loss: 0.6072 - accuracy: 0.6918 - val_loss: 0.5630 - val
_accuracy: 0.7059
Epoch 39/50
5/5 [==============================] - 4s 980ms/step - loss: 0.5836 - accuracy: 0.7055 - val_loss: 0.5712 - val
_accuracy: 0.7353
Epoch 40/50
5/5 [==============================] - 4s 983ms/step - loss: 0.6057 - accuracy: 0.7329 - val_loss: 0.6025 - val
_accuracy: 0.6765
Epoch 41/50
5/5 [==============================] - 4s 989ms/step - loss: 0.6501 - accuracy: 0.6712 - val_loss: 0.5997 - val
_accuracy: 0.6765
Epoch 42/50
5/5 [==============================] - 4s 896ms/step - loss: 0.6402 - accuracy: 0.6644 - val_loss: 0.5656 - val
_accuracy: 0.7059
Epoch 43/50
5/5 [==============================] - 5s 933ms/step - loss: 0.5961 - accuracy: 0.7055 - val_loss: 0.5980 - val
_accuracy: 0.6471
Epoch 44/50
5/5 [==============================] - 4s 911ms/step - loss: 0.5966 - accuracy: 0.7055 - val_loss: 0.5562 - val
_accuracy: 0.7059
Epoch 45/50
5/5 [==============================] - 4s 1s/step - loss: 0.5756 - accuracy: 0.7192 - val_loss: 0.5668 - val_ac
curacy: 0.6471
Epoch 46/50
5/5 [==============================] - 4s 1s/step - loss: 0.5652 - accuracy: 0.7329 - val_loss: 0.5666 - val_ac
curacy: 0.7059
Epoch 47/50
5/5 [==============================] - 4s 904ms/step - loss: 0.5799 - accuracy: 0.7397 - val_loss: 0.6453 - val
_accuracy: 0.6471
Epoch 48/50
5/5 [==============================] - 4s 907ms/step - loss: 0.5867 - accuracy: 0.6986 - val_loss: 0.6411 - val
_accuracy: 0.6471
Epoch 49/50
5/5 [==============================] - 4s 908ms/step - loss: 0.5597 - accuracy: 0.7192 - val_loss: 0.5646 - val
_accuracy: 0.7059
Epoch 50/50
5/5 [==============================] - 4s 900ms/step - loss: 0.5711 - accuracy: 0.7397 - val_loss: 0.5869 - val
_accuracy: 0.6765
```

Out[13]:

```
<keras.callbacks.History at 0x1bb59166370>
```

Evaluating the Model

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

In [14]:

```python
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 [==============================] - 3s 670ms/step - loss: 0.5492 - accuracy: 0.7397
The accurancy of the model on the training set is 73.97 %
```

In [15]:

```python
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 45ms/step - loss: 0.5869 - accuracy: 0.6765
The accurancy of the model on the test set is 67.65 %
```