Project Review 3

Team members:

Mihir Ingole
Eldho Joy
Dannasri Srinivasan

Project topic:

Classification of Autism Spectrum Disorder using computer vision with deep learning and the ABIDE dataset

Project description:

Autism spectrum disorder is a neurological disease which affects the social interactions and causes repetitive behaviors in patients. As per the data from WHO, one child in every 160 is affected by ASD worldwide also according to the report from CDC, ASD affects one in 69 children in the USA.

This project aims at identifying ASD patients using functional Magnetic Resonance Imaging data from of ASD patients from typical control patients using a Convolutional Neural Network. Diagnosing neurological diseases such as epilepsy, Alzheimer and autism requires developing a model based on functional and structural region relationships in the brain. Therefore, we are using fMRI data to detect potential biomarkers from the brain for autism. It mainly detects the correlated fluctuations in the blood oxygen level-dependent signals from the brain regions. We will be investigating objective biomarkers in the Autism Brain Imaging Data Exchange (ABIDE I) dataset.

Data preparation:

The data used in the study that we are referring for this project is the pre-processed version of Autism Brain Imaging Data Exchange (ABIDE) dataset. This dataset publicly available and can be obtained using the Functional Connectomes Project (FCP) configurable pipeline, the analysis of Connectomes (CPAC). After filtering the data as per the preprocessing requirements of the study, we obtain 871 quality fMRI images with phenotypic information.
Parcellation atlas of the brain used for this project as per the study referred is **CC400**.
The steps included in preprocessing are as follows: -

- Slice timing correction
- Correction for motion
- Normalization of voxel intensity
- Nuisance regression (employed to delete the signal fluctuations)
- Bandpass filtering (0.01–10 HZ)
- Repetition time adjustments (varies with the site)

- Echo time adjustments (varies with the site)
- T1 (tissue time)

Obtaining dataset:

We obtained dataset from the publicly available NIlearn dataset for abide. The dataset we used is Autism Brain Imaging Data Exchange (ABIDE) with CC400 parcellation atlas and we used CPAC pipeline with bandpass filtering and global signal regression to obtain the dataset along with the above mentioned pre-processing.

```
!pip install --upgrade nilearn

from nilearn.datasets import fetch_abide_pcp
# Fetch the full data and update phenotypic data and cross_validation
abide = fetch_abide_pcp(derivatives = ['rois_cc400'], pipeline = 'cpac', quality_checked = True,
                        zband_pass_filtering=True, global_signal_regression=True)

y = abide.phenotypic['DX_GROUP']
```

Specifications of resources:

- Used by our team.
    - GPU: NVIDIA GeForce RTX 3050 Ti
    - Processor: AMD Ryzen 7 5800H Processor (8 Core)
    - 16 GB DDR5 RAM and SSD
    - And Google Colab
- Used by the authors in previous research papers.
    - *NVIDIA Tesla K80 model GPU*
    - *24GB GDDR5*

Creating Correlation Matrices using PCC:

The Pearson Correlation Coefficient calculates the correlation between 2 areas of the brain and thereby presents a correlation matrix between all the 392 regions. This results in 392 by 392 matrix whose upper and lower diagonal contain the same values and each value represents the correlation between the corresponding regions of the brain. Here we are implementing the correlations with the help of "*corrcoef*" function of NumPy.

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')

abide_loaded = load("/content/drive/My Drive/Colab Notebooks/abide_dataset.npy",allow_pickle=True)
y_loaded = load('/content/drive/My Drive/Colab Notebooks/labels.npy',allow_pickle=True)
y_loaded[y_loaded==2] = 0
correlationMartices = list()

for elem in abide_loaded:
    correlationMartices.append(np.corrcoef(np.transpose(elem)))

correlationMartices = np.array(correlationMartices)
```

```
correlationMartices[0]
```

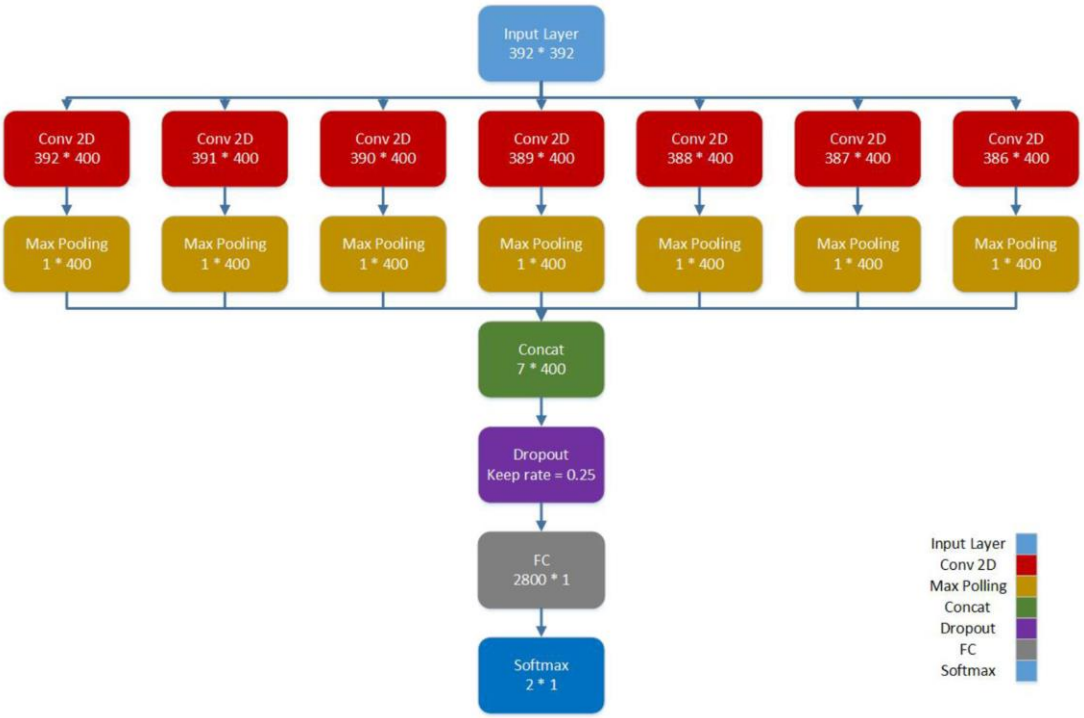
```
array([[ 1.        , -0.18410496, -0.16604868, ...,  0.10782889,
         0.13257605,  0.08509509],
       [-0.18410496,  1.        ,  0.34760289, ..., -0.10660081,
         0.0901855 , -0.23478802],
       [-0.16604868,  0.34760289,  1.        , ..., -0.29897388,
         0.02292012, -0.04428938],
       ...,
       [ 0.10782889, -0.10660081, -0.29897388, ...,  1.        ,
         0.25247047,  0.33777588],
       [ 0.13257605,  0.0901855 ,  0.02292012, ...,  0.25247047,
         1.        ,  0.20485758],
       [ 0.08509509, -0.23478802, -0.04428938, ...,  0.33777588,
         0.20485758,  1.        ]])
```

```
correlationMartices[0].shape
```

```
(392, 392)
```

Model Implementation:

The functional connectivity matrices between the pairs of ROI are fed as input to convolution layers. Our final CNN model contains 7 convolution layers with tanh activation stacked, with each layer having a vector as filters ranging from 1x392 to 7x392. We are creating 400 such filters for each layer and Max-Pooling them to dimensions of 1x400. All the concatenation after maxpooling of each layer will result in dimensions of 7x400. Further flattening and dropout is applied with dropout rate of 0.25. Two dense layers with 2800 and 2 units respectively will be added in order to determine the probability of each class with Softmax activation.



Below is the implementation using keras:-

In [5]:

```
inp = Input(shape=(392,392,1))

model1 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (1,392))(inp)
model1 = tf.keras.layers.Activation('tanh')(model1)
model1 = tf.keras.layers.MaxPooling2D(1,400)(model1)


model2 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (2,392))(inp)
model2 = tf.keras.layers.Activation('tanh')(model2)
model2 = tf.keras.layers.MaxPooling2D(1,400)(model2)


model3 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (3,392))(inp)
model3 = tf.keras.layers.Activation('tanh')(model3)
model3 = tf.keras.layers.MaxPooling2D(1,400)(model3)


model4 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (4,392))(inp)
model4 = tf.keras.layers.Activation('tanh')(model4)
model4 = tf.keras.layers.MaxPooling2D(1,400)(model4)


model5 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (5,392))(inp)
model5 = tf.keras.layers.Activation('tanh')(model5)
model5 = tf.keras.layers.MaxPooling2D(1,400)(model5)


model6 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (6,392))(inp)
model6 = tf.keras.layers.Activation('tanh')(model6)
model6 = tf.keras.layers.MaxPooling2D(1,400)(model6)


model7 = tf.keras.layers.Conv2D(filters = 400, kernel_size = (7,392))(inp)
model7 = tf.keras.layers.Activation('tanh')(model7)
model7 = tf.keras.layers.MaxPooling2D(1,400)(model7)
```

In [6]:
```
model = tf.keras.layers.concatenate([model1, model2, model3, model4, model5, model6, model7], axis=1)
model = tf.keras.layers.Dropout(0.25)(model)
model = tf.keras.layers.Flatten()(model)
model = tf.keras.layers.Dense(400, activation='tanh')(model)
model = tf.keras.layers.Dense(2, activation='softmax')(model)
model = tf.keras.Model(inputs=inp, outputs=model)
```

In [7]:
```
model.summary()
```

```
Model: "model"
```

Model Summary:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 392, 392, 1)]	0	[]
conv2d (Conv2D)	(None, 392, 1, 400)	157200	['input_1[0][0]']
conv2d_1 (Conv2D)	(None, 391, 1, 400)	314000	['input_1[0][0]']
conv2d_2 (Conv2D)	(None, 390, 1, 400)	470800	['input_1[0][0]']
conv2d_3 (Conv2D)	(None, 389, 1, 400)	627600	['input_1[0][0]']
conv2d_4 (Conv2D)	(None, 388, 1, 400)	784400	['input_1[0][0]']
conv2d_5 (Conv2D)	(None, 387, 1, 400)	941200	['input_1[0][0]']
conv2d_6 (Conv2D)	(None, 386, 1, 400)	1098000	['input_1[0][0]']

Model Hyperparameters:

- Activations: tanh and softmax
- Loss: Sparse Categorical Crossentropy
- Optimizer: Adam
- Learning Rate: 0.005
- Batch Size: 32
- Epochs: 300
- Strategy: 10-fold cross-validation

Difference in our work and reference used:

We have decided to implement a dense layer of 400 units instead of 2800 due to the computational limitations. This is not necessarily an enhancement to the model, but the changes are expected to impact the results. The positive or negative results of the impact will be analyzed in the further phases of the project.

max_pooling2d (MaxPooling2D)	(None, 1, 1, 400)	0	['activation[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_1[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_2[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_3[0][0]']
max_pooling2d_4 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_4[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_5[0][0]']
max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 400)	0	['activation_6[0][0]']
concatenate (Concatenate)	(None, 7, 1, 400)	0	['max_pooling2d[0][0]', 'max_pooling2d_1[0][0]', 'max_pooling2d_2[0][0]', 'max_pooling2d_3[0][0]', 'max_pooling2d_4[0][0]', 'max_pooling2d_5[0][0]', 'max_pooling2d_6[0][0]']
dropout (Dropout)	(None, 7, 1, 400)	0	['concatenate[0][0]']
flatten (Flatten)	(None, 2800)	0	['dropout[0][0]']
dense (Dense)	(None, 400)	1120400	['flatten[0][0]']
dense_1 (Dense)	(None, 2)	802	['dense[0][0]']

Challenges and limitations:

The reduction in the units in Dense layer from 2800 to 400 has reduced the number of trainable parameters by more than half, thereby improving the training efficiency.

```
==================================
Total params: 5,514,402
Trainable params: 5,514,402
Non-trainable params: 0
__________________________________
==================================
Total params: 12,241,602
Trainable params: 12,241,602
Non-trainable params: 0
__________________________________
```

But this is still resulting in less significant reduction in the training loss as shown below.

```
25/25 [==============================] - 104s 4s/step - loss: nan - accuracy: 0.5402
3/3 [==============================] - 5s 1s/step - loss: nan - accuracy: 0.5057
Epoch 1/300
25/25 [==============================] - 104s 4s/step - loss: nan - accuracy: 0.5287
Epoch 2/300
25/25 [==============================] - 102s 4s/step - loss: nan - accuracy: 0.5287
Epoch 3/300
25/25 [==============================] - 104s 4s/step - loss: nan - accuracy: 0.5287
Epoch 4/300
25/25 [==============================] - 104s 4s/step - loss: nan - accuracy: 0.5287
Epoch 5/300
25/25 [==============================] - 102s 4s/step - loss: nan - accuracy: 0.5287
Epoch 6/300
25/25 [==============================] - ETA: 0s - loss: nan - accuracy: 0.5287
Loss no longer changing. Hence stopping the training!

25/25 [==============================] - 103s 4s/step - loss: nan - accuracy: 0.5287
3/3 [==============================] - 5s 1s/step - loss: nan - accuracy: 0.6092
```

Implementing 10-Fold Cross validation:

In [9]:

```
accuracy_metrics = list()

for ifold in range(len(datasetWithKFolds)):
    callbacks = myCallbacks()

    completeDataset = list(datasetWithKFolds)
    testFold = completeDataset.pop(ifold)

    trainFolds = list()
    for elem in completeDataset:
        trainFolds += elem

    data_test = list()
    labels_test = list()
    for testElem in testFold:
        data_test.append(testElem[0])
        labels_test.append(testElem[1])

    data_train = list()
    labels_train = list()
    for trainElem in trainFolds:
        data_train.append(trainElem[0])
        labels_train.append(trainElem[1])

    data_train = np.array(data_train)
    labels_train = np.array(labels_train)
    data_test = np.array(data_test)
    labels_test = np.array(labels_test)

    model.fit(x=data_train,y=labels_train,batch_size=32,epochs=300, callbacks=callbacks)
    accuracy_metrics.append(model.evaluate(data_test, labels_test)[1])

accuracy = sum(accuracy_metrics)/len(accuracy_metrics)
print("Final model accuracy: ",accuracy)
```

Compilation and Accuracy:

```
In [8]: 1 model.compile(optimizer=keras.optimizers.SGD(learning_rate=0.005), loss='sparse_categorical_crossentropy',
        2               metrics=['accuracy'])
        3 #model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(), optimizer=keras.optimizers.Adam(learning_rate=
```

```
25/25 [==============================] - 104s
3/3 [==============================] - 5s 1s/
Final model accuracy:  0.5367816120386124
```

Accuracy on best fold:

```
ss: nan - accuracy: 0.5287
ıan - accuracy: 0.6092
```

References:

Automated Detection of Autism Spectrum Disorder Using a Convolutional Neural Network [Zeinab Sherkatghanad, Mohammadsadegh Akhondzadeh, Soorena Salari, Mariam Zomorodi-Moghadam, Moloud Abdar, U. Rajendra Acharya, Reza Khosrowabadi and Vahid Salari.]

Identification of autism spectrum disorder using deep learning and the ABIDE dataset [Anibal Sólon Heinsfeld, Alexandre Rosa Franco R. Cameron Craddock, Augusto Buchweitz, Felipe Meneguzzi]

Multisite functional connectivity MRI classification of autism: ABIDE results [JaredA.Nielsen, Brandon A. Zielinski, P. Thomas Fletcher, Andrew L. Alexander, NicholasLange, ErinD.Bigler, Janet E. Lainhart and Jeffrey S. Anderson]