DataScientest

Report 1: exploration, data visualization and data pre-processing report

Introduction to the project

Context

Computer-assisted diagnostic of psychiatric disorders have gained increasing interest over the last decade. The data often used in those tools is structural Magnetic Resonance Imaging (sMRI) and standardized clinical ratings. However, accuracies e.g., in the classification between patient and non-patient range only between 65-75% when combining clinical and sMRI data and even lower accuracies when using sMRI data only. One reason for low accuracies is, that psychiatric disorders do not manifest in alterations of single brain structures like it is the case in oncological diseases. Another reason, in return, might be the machine-learning approach used for classification between disease and healthy and the (relatively) small sample sizes of clinical populations.

To overcome these weaknesses, deep learning approaches have been introduced. Deep Learning is a specific subfield of machine learning, emphasizing the learning of successive layers to increase the meaningfulness of the output. The number of layers represents the depth of the model. The idea is, that the layers iteratively extract features from the input data. For example, in image processing, first layers may identify edges, while later layers may extract concepts relevant to a human such as digits or letters or faces.

The transformation implemented by a layer to predict the output data is parameterized by its weights. Thus, the learning process consists of finding the set of values for the weights which leads to the most accurate prediction. To determine how far the output is from the true target, a loss function is calculated based on distance measures. The basic mechanism in deep learning is to use this score as feedback to adjust the value of the weights in order to minimize the loss score. If the loss score is at a minimum, the model's best fit has been reached. The adaptation is the task of the optimizer, who implements the backpropagation algorithm, which is the central algorithm in deep learning (**see figure 1**).

In the last decade, abundant work has been made in the field of computer vision in general as well as in the field of medical image analysis in specific. For example, convolutional neural networks (cnn) have been introduced for imaging techniques (i.e., ultrasound, endoscopy, skeletal system X-rays, Breast X-Rays, and Brain MRI) as well as for human body systems (breast, eyes, skin, tooth, lungs and brain). These models are trained with huge sample sizes on non-medical images, their use on human medical images, however, has been proven adequate by collaborations between medical experts and leaders in deep learning. Thus, applying pre-trained weights in the development of new models has become state of the art today.

Transfer learning is recommended in medical image analyses as sample sizes usually are relatively small and transfer learning is one way to overcome this limitation and, therefore, avoid overfitting. Specifically, in the case of computer vision, many pretrained models are publicly available for download and can be used to boot-strap powerful vision models out of very little data. This is one of the greatest strengths of deep learning: the reuse of pre-trained weights.
To date both, deep learning and transfer learning have predominantly been used in the prediction of developing Alzheimer's Disease (Afzal et al., 2019; Maqsood et al., 2019), which is, by the way, the only psychiatric disease with well described brain manifestation in terms of (regionally developing) cortical atrophy. Considering the publication bias, that only good results are being published in medical journals, it might reflect, that analyses have also been performed in several psychiatric diseases, however, with accuracies not worth to publish.

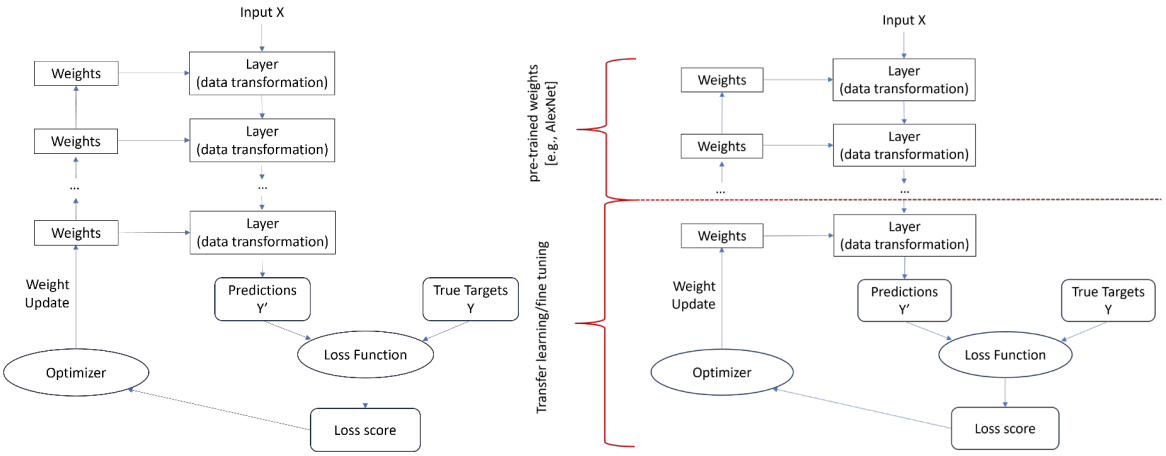

Figure 1. A neural network and the process of deep learning (left) and transfer learning (right)

Objectives

To address the question, whether insufficient accuracies in psychiatric disorders are due to inadequate machine-learning procedures or based on unsystematic brain manifestation of the disease, deep learning and transfer learning will be performed at the example of Attention-Deficit- Hyperactivity Disorder (ADHD).
Furthermore, the current project focusses on the potential of sMRI data in the prediction of psychiatric disorders. sMRI is a standard sequence in the clinical imaging protocol with comparable sequence parameters across scanners worldwide. As no active participation of the patients during data acquisition is required, sMRI data can be anonymized, shared and evaluated similarly across sites worldwide and models can be developed across ethnicities in terms of globally valid predictions. Precise illness prediction shortens the diagnostic process and would save time for the patients and costs for the health systems.

Susanne Neufang: ~20yrs of experience of analyzing neuroimaging, 12yrs of ADHD research and 5yrs of using (matlab-based) multi-model ML using NeuroMiner in the context of schizophrenia (https://www.pronia.eu/, http://www.proniapredictors.eu/neurominer/index.html)

In the context of this project, I contacted Prof. Barbara Franke from Radboud University in her position of head of the ENIGMA ADHD working group (https://enigma-brain.org/main/index.html%3Fp=1622.html). She told me, that they are currently working on DL-based classifiers, however, using cortical volume scores calculated with freesurfer (i.e. a 2D data matrix). She made contact with the dedicated co-worker and is very interested in the results. After a first exchange of project details, the correspondence with the co-worker ended.

Understanding and manipulation of data

Framework

The train and test set involved 395 children and adolescents with and without ADHD (N_{ADHD}=266, N_{hCon}=129 with hCon for healthy control subjects). It is a subsample from the Rutgers University sample of the Human Brain Network data set (http://fcon_1000.projects.nitrc.org/indi/cmi_healthy_brain_network/). For external validation, a data set of 136 ADHD patients and hCon are analyzed (N_{ADHD}=61, N_{hCon}=75), collected inhouse in a project funded by the German Research association (DFG, Gran No 454502177).
sMRI data acquisition was performed on a 3 Tesla in terms of an isotropic high-resolution T1-weighted three-dimensional structural image (magnetization prepared rapid gradient echo, 176 slices, 1×1×1mm^3, repetition time=2400ms, echo time=2.26ms, field of view=256mm^2, flip angle=9°).

Relevance

From earlier findings, structural alterations associated with ADHD have been in fronto-parietal regions, subcortical structures such as the striatum as well as the cerebellum. Therefore, whole brain analyses were performed, in both, the 3D and the 2D approach.
Binary classifications were performed with the target variable ADHD (1) vs. hCon (0). Limitations were based on sample size due to bad data quality of the HBN data set.

Pre-processing and feature engineering

Before entering the ML-processing, data was skull-stripped and registered to the MNI (https://www.bic.mni.mcgill.ca/ServicesAtlases/ICBM152NLin2009) standard

space as implemented in the SynthStrip workflow (https://surfer.nmr.mgh.harvard.edu/docs/synthstrip/) of the freesurfer software package (https://freesurfer.net/).
For the 3D approach, the resulting 3D nii-files were used. For the 2D approach, the files were split into axial slices under the consideration of the anatomical aspect ratio (https://ianmcatee.com/converting-a-nifti-file-to-an-image-sequence-using-python/, s. code-file NIfTI_2_ImageSequence.ipynb). The 30 central axial slices entered the analysis.
In both approaches, 2D and 3D data was normalized to correct for intensity variations across images via min max normalization. In the current analyses, no dimensionality reduction (such as PCA) has been performed as the present data sets were not extensive. However, this issue has been addressed in the Continuation of the project section of this report.

Visualizations and Statistics

For all analyses, please see code Vis_apriori_Train.ipynb.
In the training sample, the ratio between patients and hCon were 67:32%.

The role of age

Subjects were aged between 6 and 20yrs, with an average age of M_{age}=10.9± 3.2 yrs. Age did not differ significantly between ADHD patients and hCon (M_{age_ADHD}=10.9± 3.1yrs, $M_{agehCon}$=10.8± 3.3yrs, $T_{(393,2)}$= 0.4 and p= 0.689). As the data set consists of developing brain images, age distribution was plotted group-specifically showing comparable distributions between groups (see **figure 3**).

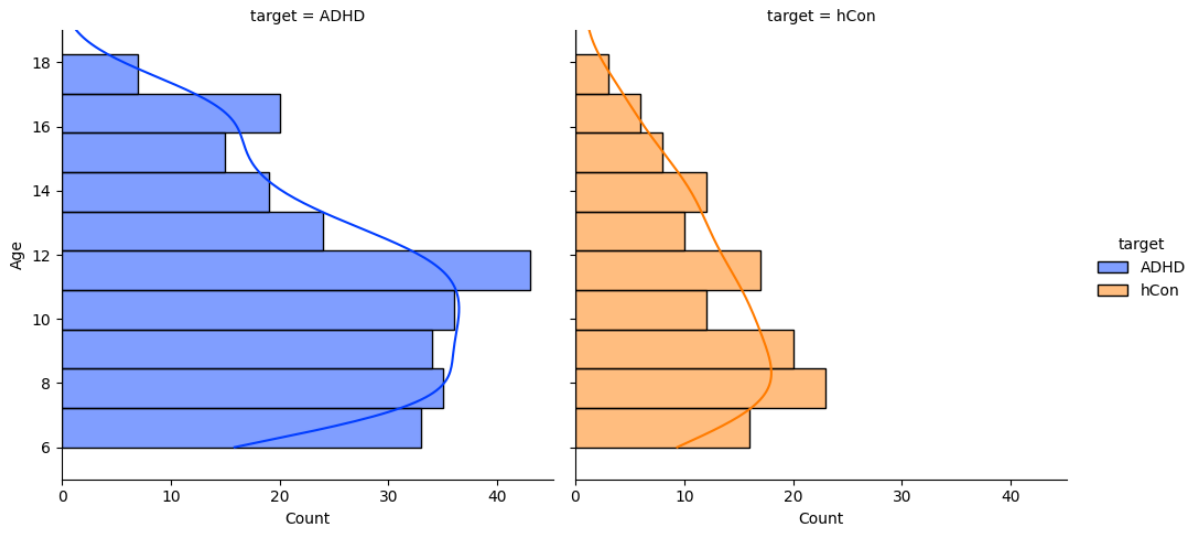

Figure 2. On the left, age distribution of ADHD patients is presented, showing that the majority is between 6 and 11 years old. On the right, the age distribution of healthy controls shows a similar curve.

The role of Sex

Across all training subjects the sex distribution is 69.3% males and 30.6% females, in the patient group sex ratio was 78.2:21.8%, in the healthy control group 51.16:48.84%. Using a Chi-Square Test, it was revealed, that the sex ratio differs significantly between patients and hCon ($Male_{ADHD}$=208, $Female_{ADHD}$= 58, $Male_{hCon}$=68, $Female_{hCon}$=61, $X_{(394,1)}$= 28.6 and p=.000, see **figure 4**) .

The higher proportion of male patients reflects the real-world situation with an estimated male:female ratio of 2.3:1 in the US (in Germany 5-9times more often in males compared to females). Thus, this effect might influence the analyses, however, increases their ecological validity.

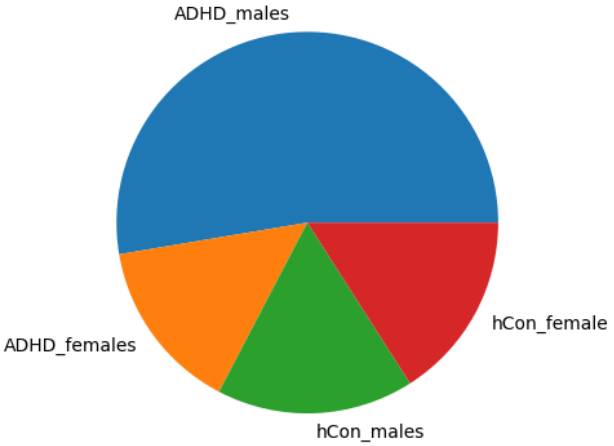

Figure 3 group-specific sex distribution.

The role of Total Intracranial Volume (TIV)

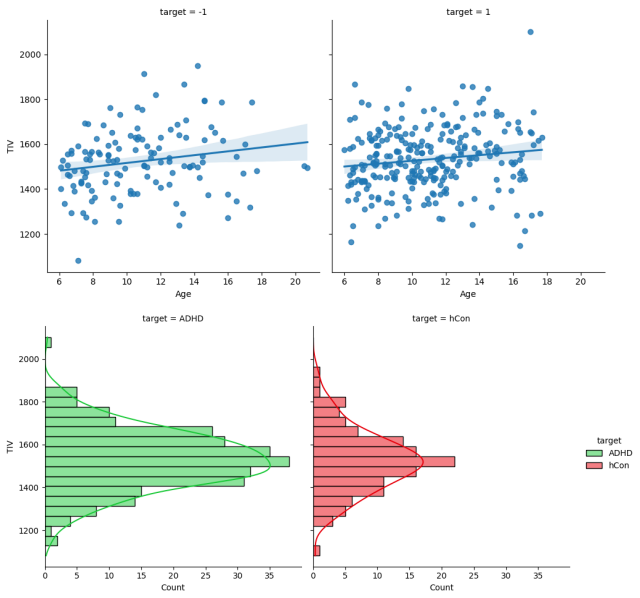

Figure 4. In the upper row, group-specific development of brain volume is plotted, in the lower row the distribution of TIV for both groups.

TIV was comparable between groups (M_{TIV_ADHD}=1532 ± 138mm³, M_{TIV_hCon}=1523 ± 147mm³, $T_{(393,2)}$= 0.6, p= 0.578) and also similarly distributed. Additionally, TIV increased with age (R=0.16, p=0.001), with brain development not differing significantly between ADHD patients and hCon (see **figure 5**).

In all, *a priori* effects were expected and reflect the real-world scenario. In the current analyses, variables did not enter as nuisance variables in the analyses. However, this point will also be addressed in the in the Continuation of the project section of the report.

DataScientest

Report 2: modeling report

Stages of the project

Classification of the problem

In the current project, a classification between ADHD patients and healthy controls has been performed based on 3D sMRI data. It is a computer vision task using deep learning and transfer learning. Training results were presented in terms of model accuracy, prediction results additionally in terms of confusion matrix, Precision, Recall, F1-Score and Area under the curve (AUC).

Model choice and optimization

Using deep learning on sMRI data, two approaches have been suggested: a 3D approach loading 3D as nifti-files into the model, and a 2D approach, splitting the data into slices of png-data format before loading the data for analyses. In this project, both methods have been performed. Deep/Transfer Learning was applied as data was imaging data with Computer Vision being the most adequate analysis approach.

Table 1 Model definition

	Preprocessing	algorithms	Parameter optimization
2D	Feat*_center Feat*_std_norm	2D CNN	ResNet50 InceptionV3
3D	MinMax norm resize	3D CNN	Keras.optimizers.schedules.ExponentialDecay

Interpretation of results

Both models, 2D and 3D stopped early based on lack of acc improvement. In both models, training accuracy varied around 60%. Varying pre-trained models in the 2D approach did not change accuracy considerably, neither did varying between linear and sigmoid activation functions in the output layers.

Table 2 Results of model training

	Acc	Loss	Val_acc	Val_loss
2D	0.609	0.661	0.574	0.683
3D	0.619	0.692	0.564	0.697

Based on the poor results of the models, no interpretability technique has been applied. However, as model interpretation plays a crucial role in medical data science (Clinicians need to know, what parameters were crucial for the clinical

question to increase the focus on the same), as a future step, a hybrid learning approach is planned to combine feature extraction from pre-trained model with SVM and SHAP for the quantification of feature importance. Additionally, the application of pre-trained 3D models is planned using e.g. AlexNet3D (https://github.com/denti/AlexNet3D) for deep as well as hybrid models.

Final report :

Conclusion drawn

Difficulties encountered during the project

The main obstacle of this project is based on the nature of the data, or better the lack of structural manifestation of ADHD in the brain. With only small and unsystematic brain alterations in ADHD patients, any algorithm is challenged to find differences. The second point is the small sample sizes- even though we performed transfer learning in the 2D approach to overcome this issue, training results remain poor.

- Forecast: Data preprocessing outside the ML-framework (data inspection, data quality assessment, data slicing etc) took more time than expected
- Datasets: data acquisition had already been performed, based on 3D nii-files, there were no missing values or duplicates. Additionally, no change in data-types had to be performed, so data cleaning was not necessary.
- Technical/theoretical skills: the acquisition of deep learning skills took a lot of time and the completion of many modules/exercises within the platform.
- Relevance: The project has high clinical relevance in terms of decrease of the time of diagnostical process.
- IT: Using a high-performance PC, I did not run into problems.

Report

The main contribution of this project is the introduction of two 2D and 3D deep learning workflows for the analysis of medical data in general. However, at the example of the psychiatric disorder ADHD, accuracies of the models were not convincing. This finding is supported by the lack of publications on this issue.

Table 3 prediction results

	TP	FP	TN	FN	Acc	Precision	Recall	F1-score	AUC
2D	2243	7	4	1826	0.551	0.36	0.002	0	0.489
3D	19	9	52	56	0.522	0.48	0.85	0.61	0.487

The goals of the project were to develop a deep learning framework for the analysis of sMRI data. This goal has been achieved, as two different approaches have been developed, one 2D and one 3D. However, as results were very poor, the questions remain, whether (a) the codes contain major errors and do not match the input data or (b) sMRI is not apt for the classification between ADHD patients and healthy controls as data does not contain systematic differences between classes and there was not much to detect. To check whether one or both aspects are present, the codes should be tested on sMRI data with manifest structural changes (such as tumors).

Continuation of the project

Additional steps in data preprocessing

- include dimensionality reduction
- include regression for nuisance variables as age and sex (esp. age as it is a developmental sample)

Pre-trained models

- test further pretrained models in the 2D approach
- add pre-trained models in the 3D approach
- use pre-trained models only for feature extraction and perform e.g. SVM (hybrid approach)

Model Interpretability

- add methods of model-interpretability to identify key regions

Findings of the project have shown the challenges of using deep learning on sMRI (only) as computer-assisted diagnostic tool.

Bibliography

(https://ianmcatee.com/converting-a-nifti-file-to-an-image-sequence-using-python/

Appendices

Codes and data are available in Github
(https://github.com/SusuNeu/project_ds_sep23)

Code for 2D analysis (project_2D.ipynb)

In [ ]:
```
# ------------------------------------------------------------------------
#                        Import libraries
# ------------------------------------------------------------------------


from keras.models import load_model
from keras.layers import Activation,Dense
from keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix,precision_sco
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt_False_Positive_vs_True_Positive

from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
```

In [ ]:
```
# ------------------------------------------------------------------------
#                   initialization of pre-trained model
# ------------------------------------------------------------------------

# #(a) ResNet

# def Build_CNN_Model():
#     # load ResNet model
#     ResNet = ResNet50(include_top=True, input_shape=(224, 224, 3))

#     # mark loaded layers as not trainable
#     for layer in ResNet.layers:
#         layer.trainable = False

#     # mark loaded layers as trainable
#     for layer in ResNet.layers[100:]:
#         layer.trainable = True

#    #  Softmax Classifier
#     Class_layer = Dense(2)(ResNet.layers[-2].output)

#     Softmax_layer = Activation('softmax')(Class_layer)

#     # define new model
#     model = Model(inputs=ResNet.inputs, outputs=Softmax_layer)

#     #  Display model
#     model.summary()

#     # compile model

#     opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
#     model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accur

#     return model
```

In [ ]:
```
#(b) InceptionV3

def Build_CNN_Model():

    # load Inception model
    Inception = InceptionV3(include_top=True, input_shape=(299, 299, 3))

    # mark loaded layers as trainable
    for layer in Inception.layers:
        layer.trainable = True

    #  Softmax Classifier
    Class_layer = Dense(2)(Inception.layers[-2].output)

    Softmax_layer = Activation('softmax')(Class_layer)


    # define new model
    model = Model(inputs=Inception.inputs, outputs=Softmax_layer)


    #  Display model
    model.summary()

    # compile model
    opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accurac

    return model
```

In [ ]:
```
# ------------------------------------------------------------------------
#                          model training
# ------------------------------------------------------------------------

def Train_CNN_Model(model):
    # create data generators
    train_datagen = ImageDataGenerator(
                                       rescale=1.0/255.0,
                                       featurewise_center= True,
                                       featurewise_std_normalization = True,
                                       rotation_range=10,
                                       width_shift_range=0.1,
                                       height_shift_range=0.1,
                                       zoom_range=0.2,
                                       brightness_range=[0.2,1.0],
                                       )
    valid_datagen = ImageDataGenerator(
                                       rescale=1.0/255.0,
                                       featurewise_center= True,
                                       featurewise_std_normalization = True)



    # prepare iterators
```

```
    batch_size=4
    train_it = train_datagen.flow_from_directory('Train_2D_finalSamples/',classes =
    valid_it = valid_datagen.flow_from_directory('Valid_2D_finalSamples/',classes =


    epochs=20;

    history = model.fit(train_it, validation_data=valid_it, epochs=epochs, verbose=


    #  "Accuracy"
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

    # "Loss"
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

    # save model
    model.save('cnn_incept_2D.h5')                    # alternative format model.
```

In [ ]:

```
 # ------------------------------------------------------------------------
#                      function for model evaluation
# ------------------------------------------------------------------------


def Evaluate_CNN_Model():
    # load model
    model = load_model('cnn_incept_2D.h5')

    # load test data
    batch_size=4
    test_datagen = ImageDataGenerator(
                                      rescale=1.0/255.0,
                                      featurewise_center= True,
                                      featurewise_std_normalization = True)

    test_it = test_datagen.flow_from_directory('Test_2D_finalSamples/',classes =('h
                                           shuffle=False,batch_size=batch_size,

    y_true = test_it.classes;

    y_pred = model.predict(test_it, steps=len(test_it), verbose=1)
```

```
    y_pred_prob = y_pred[:,1]


    y_pred_binary =  y_pred_prob > 0.5

    #Confution Matrix
    print('\nConfusion Matrix\n -------------------------')
    print(confusion_matrix(y_true,y_pred_binary));

    # accuracy: (tp + tn) / (p + n)
    accuracy = accuracy_score(y_true, y_pred_binary)
    print('Accuracy: %f' % accuracy)


    # precision tp / (tp + fp)
    precision = precision_score(y_true, y_pred_binary)
    print('Precision: %f' % precision)

    # recall: tp / (tp + fn)
    recall = recall_score(y_true, y_pred_binary)
    print('Recall: %f' % recall)

    # f1: 2 tp / (2 tp + fp + fn)
    f1 = f1_score(y_true, y_pred_binary)
    print('F1 score: %f' % f1)

    # ROC AUC
    auc = roc_auc_score(y_true, y_pred_prob)
    print('ROC AUC: %f' % auc)


    # calculate roc curves
    fpr, tpr, _ = roc_curve(y_true, y_pred_prob)

    # plot the roc curve for the model
    plt.figure()
    plt_False_Positive_vs_True_Positive.plot(fpr, tpr, linestyle='--', label='')

    # axis labels
    plt_False_Positive_vs_True_Positive.xlabel('False Positive Rate')
    plt_False_Positive_vs_True_Positive.ylabel('True Positive Rate')

    # show the legend
    plt_False_Positive_vs_True_Positive.legend()
    # show the plot
    plt_False_Positive_vs_True_Positive.show()
```

In [ ]:
```
# main entry

model = Build_CNN_Model()
```

In [ ]:
```
Train_CNN_Model(model)
```

In [ ]:

```
Evaluate_CNN_Model()
```

Code for 3D analysis (project_3D.ipynb)

In [ ]:
```
# ----------------------------------------------------------------------
#                         Import libraries
# ----------------------------------------------------------------------


import os
import zipfile
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

import keras
from keras import layers
import nibabel as nib
import random

from scipy import ndimage, stats
```

In [ ]:
```
# ----------------------------------------------------------------------
#                          data load and preprocessing
# ----------------------------------------------------------------------

# # Define functions

def read_nifti_file(filepath):
    """Read and load volume"""
    # Read file
    scan = nib.load(filepath)
    # Get raw data
    scan = scan.get_fdata()
    return scan



def normalize(volume):
    """Normalize the volume"""
    volume = stats.zscore(volume, axis=None)
    minV = volume.min()
    maxV = volume.max()
    # volume[volume < minV] = minV
    # volume[volume > maxV] = maxV
    volume = (volume - minV) / (maxV - minV)
    volume = volume.astype("float32")
    return volume



def resize_volume(img):
    """Resize across z-axis"""
    # Set the desired depth
    desired_depth = 64
    desired_width = 128
    desired_height = 128
    # Get current depth
```

```
    current_depth = img.shape[-1]
    current_width = img.shape[0]
    current_height = img.shape[1]
    # Compute depth factor
    depth = current_depth / desired_depth
    width = current_width / desired_width
    height = current_height / desired_height
    depth_factor = 1 / depth
    width_factor = 1 / width
    height_factor = 1 / height
    # Rotate
    img = ndimage.rotate(img, 90, reshape=False)
    # Resize across z-axis
    img = ndimage.zoom(img, (width_factor, height_factor, depth_factor), order=1)
    return img



def process_scan(path):
    """Read and resize volume"""
    # Read scan
    volume = read_nifti_file(path)
    # Normalize
    volume = normalize(volume)
    # Resize width, height and depth
    volume = resize_volume(volume)
    return volume
```

In [ ]:

```
# # define data paths

# Folder "Train_3D_finalSamples/hCon_finalSample" contains nifti-files of healthy c
normal_scan_paths = [
    os.path.join(os.getcwd(), "Train_3D_finalSamples/hCon_finalSample", x)
    for x in os.listdir("Train_3D_finalSamples/hCon_finalSample")
]

# Folder "Train_3D_finalSamples/ADHD_finalSample" contains nifti-files of ADHD pati
abnormal_scan_paths = [
    os.path.join(os.getcwd(), "Train_3D_finalSamples/ADHD_finalSample", x)
    for x in os.listdir("Train_3D_finalSamples/ADHD_finalSample")
]

print("hCon scans 4 train: " + str(len(normal_scan_paths)))
print("ADHD scans 4 train: " + str(len(abnormal_scan_paths)))
```

In [ ]:

```
# # Train Test split

# Each scan is resized across height, width, and depth and rescaled.
abnormal_scans = np.array([process_scan(path) for path in abnormal_scan_paths])
normal_scans = np.array([process_scan(path) for path in normal_scan_paths])

# Scans of patients were assigned to label 1, healthy controls to label 0
abnormal_labels = np.array([1 for _ in range(len(abnormal_scans))])
normal_labels = np.array([0 for _ in range(len(normal_scans))])
```

```
# Split data in the ratio 70-30 for training and validation.
x_train = np.concatenate((abnormal_scans[:70], normal_scans[:56]), axis=0)
y_train = np.concatenate((abnormal_labels[:70], normal_labels[:56]), axis=0)
x_val = np.concatenate((abnormal_scans[70:], normal_scans[56:]), axis=0)
y_val = np.concatenate((abnormal_labels[70:], normal_labels[56:]), axis=0)
print(
    "Number of samples in train and validation are %d and %d."
    % (x_train.shape[0], x_val.shape[0])
)
```

In [ ]:

```
# load test data
# Folder "Test_3D_sample/ad" contains nifti-files of ADHD patients of an independen
test_ADHD_scan_paths = [
    os.path.join(os.getcwd(), "Test_3D_sample/ad", x)
    for x in os.listdir("Test_3D_sample/ad")
]
# Folder "Test_3D_sample/hc" contains nifti-files of ADHD patients of an independen
test_hCon_scan_paths = [
    os.path.join(os.getcwd(), "Test_3D_sample/hc", x)
    for x in os.listdir("Test_3D_sample/hc")
]

test_ADHD_scans = np.array([process_scan(path) for path in test_ADHD_scan_paths])
test_hCon_scans = np.array([process_scan(path) for path in test_hCon_scan_paths])

test_ADHD_labels = np.array([1 for _ in range(len(test_ADHD_scans))])
test_hCon_labels = np.array([0 for _ in range(len(test_hCon_scans))])


x_test = np.concatenate((test_ADHD_scans, test_hCon_scans), axis=0)
y_test = np.concatenate((test_ADHD_labels, test_hCon_labels), axis=0)
```

In [ ]:
```
# ----------------------------------------------------------------------
#                           data augmentation
# ----------------------------------------------------------------------

# # definition of functions

def rotate(volume):
    """Rotate the volume by a few degrees"""

    def scipy_rotate(volume):
        # define some rotation angles
        angles = [-20, -10, -5, 5, 10, 20]
        # pick angles at random
        angle = random.choice(angles)
        # rotate volume
        volume = ndimage.rotate(volume, angle, reshape=False)
        volume[volume < 0] = 0
        volume[volume > 1] = 1
        return volume

    augmented_volume = tf.numpy_function(scipy_rotate, [volume], tf.float32)
    return augmented_volume


def train_preprocessing(volume, label):
    """Process training data by rotating and adding a channel."""
    # Rotate volume
    volume = rotate(volume)
    volume = tf.expand_dims(volume, axis=3)
    return volume, label


def validation_preprocessing(volume, label):
    """Process validation data by only adding a channel."""
    volume = tf.expand_dims(volume, axis=3)
    return volume, label
```

In [ ]:
```
# Define data loaders.
train_loader = tf.data.Dataset.from_tensor_slices((x_train, y_train))
validation_loader = tf.data.Dataset.from_tensor_slices((x_val, y_val))
test_loader = tf.data.Dataset.from_tensor_slices((x_test, y_test))

batch_size = 2
# Augment the on the fly during training.
train_dataset = (
    train_loader.shuffle(len(x_train))
    .map(train_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
# Only rescale.
validation_dataset = (
    validation_loader.shuffle(len(x_val))
    .map(validation_preprocessing)
    .batch(batch_size)
```

```
    .prefetch(2)
)

test_dataset = (
    test_loader.shuffle(len(x_test))
    .map(validation_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
```

In [ ]:
```
# ------------------------------------------------------------------------
#                            data visualization
# ------------------------------------------------------------------------


#import matplotlib.pyplot as plt
# plot one slice

data = train_dataset.take(1)
images, labels = list(data)[0]
images = images.numpy()
image = images[0]
print("File dimensions are:", image.shape)
plt.imshow(np.squeeze(image[:, :, 30]), cmap="gray");
```

In [ ]:
```
# plot series of slices

def plot_slices(num_rows, num_columns, width, height, data):
    """Plot a montage of 20 sMRI slices"""
    data = np.rot90(np.array(data))
    data = np.transpose(data)
    data = np.reshape(data, (num_rows, num_columns, width, height))
    rows_data, columns_data = data.shape[0], data.shape[1]
    heights = [slc[0].shape[0] for slc in data]
    widths = [slc.shape[1] for slc in data[0]]
    fig_width = 12.0
    fig_height = fig_width * sum(heights) / sum(widths)
    f, axarr = plt.subplots(
        rows_data,
        columns_data,
        figsize=(fig_width, fig_height),
        gridspec_kw={"height_ratios": heights},
    )
    for i in range(rows_data):
        for j in range(columns_data):
            axarr[i, j].imshow(data[i][j], cmap="gray")
            axarr[i, j].axis("off")
    plt.subplots_adjust(wspace=0, hspace=0, left=0, right=1, bottom=0, top=1)
    plt.show()


# apply function for 4 rows and 10 columns for 40 slices of the sMRI scan.

plot_slices(4, 10, 128, 128, image[:, :, :40])
```

In [ ]:
```
# ---------------------------------------------------------------------
#                              data modeling
# ---------------------------------------------------------------------


# Model definition

def get_model(width=128, height=128, depth=64):
    """Build a 3D convolutional neural network model."""

    inputs = keras.Input((width, height, depth, 1))

    x = layers.Conv3D(filters=64, kernel_size=3, activation="relu")(inputs)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=64, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=128, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=256, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.GlobalAveragePooling3D()(x)
    x = layers.Dense(units=512, activation="relu")(x)
    x = layers.Dropout(0.3)(x)

    outputs = layers.Dense(units=1, activation="sigmoid")(x)

    # Define input and output of the model.
    model = keras.Model(inputs, outputs, name="3dcnn")
    return model


# Build model.
model = get_model(width=128, height=128, depth=64)
model.summary()
```

In [ ]:
```
# ---------------------------------------------------------------------
#                          Model Training
# ---------------------------------------------------------------------


# Compile model.
initial_learning_rate = 0.0001
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100000, decay_rate=0.96, staircase=True
)
model.compile(
    loss="binary_crossentropy",
```

```
    optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
    metrics=["acc"],
    run_eagerly=True,
)

# Define callbacks.
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    "3d_image_classification.keras", save_best_only=True
)
early_stopping_cb = keras.callbacks.EarlyStopping(monitor="val_acc", patience=15)

# Train the model, doing validation at the end of each epoch
epochs = 100
cnn3d_hist = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    shuffle=True,
    verbose=2,
    callbacks=[checkpoint_cb, early_stopping_cb],
)
```

In [ ]:
```
# ------------------------------------------------------------------------
#                          Model Evaluation
# ------------------------------------------------------------------------

test_pred = model.predict(test_dataset)
y_pred_binary =  test_pred > 0.5
y_true = y_test
y_pred_prob = test_pred
```

In [ ]:
```
from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix,precision_sco
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt_False_Positive_vs_True_Positive

    #Confution Matrix
print('\nConfusion Matrix\n -------------------------')
print(confusion_matrix(y_true,y_pred_binary));

    # accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_true, y_pred_binary)
print('Accuracy: %f' % accuracy)


    # precision tp / (tp + fp)
precision = precision_score(y_true, y_pred_binary)
print('Precision: %f' % precision)

    # recall: tp / (tp + fn)
recall = recall_score(y_true, y_pred_binary)
print('Recall: %f' % recall)

    # f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_true, y_pred_binary)
print('F1 score: %f' % f1)
```

```
    # ROC AUC
auc = roc_auc_score(y_true, y_pred_prob)
print('ROC AUC: %f' % auc)


    # calculate roc curves
fpr, tpr, _ = roc_curve(y_true, y_pred_prob)

    # plot the roc curve for the model
plt.figure()
plt_False_Positive_vs_True_Positive.plot(fpr, tpr, linestyle='--', label='')

    # axis labels
plt_False_Positive_vs_True_Positive.xlabel('False Positive Rate')
plt_False_Positive_vs_True_Positive.ylabel('True Positive Rate')

    # show the legend
plt_False_Positive_vs_True_Positive.legend()
    # show the plot
plt_False_Positive_vs_True_Positive.show()
```