Contents

List of Figures

ABSTRACT

Brain decoding from functional Magnetic Resonance Imaging (fMRI) represents a transformative approach in cognitive neuroscience for understanding neural mechanisms underlying human emotions. This research presents a deep learning-based brain decoding system that automatically recognizes emotional states from fMRI recordings. We develop a convolutional neural network (CNN) architecture specifically designed for functional connectivity matrix classification, integrating features from three complementary brain parcellation atlases (Harvard-Oxford, AAL, and Destrieux). The deep learning model processes connectivity patterns as 2D images, learning hierarchical representations directly from data. Validation on the ds003548 emotional faces dataset containing 16 subjects and four emotion categories demonstrates that our CNN-based brain decoding system achieves 84% classification accuracy, significantly outperforming traditional machine learning approaches (68-76% accuracy). The deep learning framework automatically discovers discriminative connectivity patterns without manual feature engineering, with Leave-One-Subject-Out cross-validation confirming robust generalization to unseen individuals. Our results establish deep learning as a superior approach for brain decoding applications, with implications for brain-computer interfaces, affective disorder diagnosis, and cognitive neuroscience research.

Chapter 1

INTRODUCTION

1.1 Overview

Brain decoding represents a transformative paradigm in computational neuroscience, enabling direct inference of mental states, cognitive processes, and emotional experiences from patterns of neural activity recorded through neuroimaging technologies. The ability to decode human emotions from functional Magnetic Resonance Imaging (fMRI) data has emerged as a critical challenge with far-reaching implications across multiple domains including clinical psychiatry, human-computer interaction, neuroscience research, and assistive technology development.

Functional MRI provides non-invasive measurements of brain activity by detecting blood-oxygen-level-dependent (BOLD) signal changes, offering unprecedented windows into the neural mechanisms underlying emotional processing. However, the complexity of brain activity patterns, high dimensionality of neuroimaging data, and substantial individual variability present significant analytical challenges for traditional pattern recognition approaches.

1.2 Motivation

The motivation for developing advanced brain decoding systems stems from several critical factors:

1.2.1 Clinical Applications

Mental health disorders affect millions globally, with conditions such as depression, anxiety, and post-traumatic stress disorder often involving dysfunctional emotional processing. Current diagnostic ap-

proaches rely heavily on subjective self-reporting and behavioral observations, lacking objective neurobiological markers. Brain decoding systems capable of identifying emotional states from fMRI data could provide:

- Objective diagnostic biomarkers for affective disorders
- Treatment response monitoring through neural activity patterns
- Personalized intervention planning based on individual brain signatures
- Early detection of emotional processing abnormalities

1.2.2 Brain-Computer Interface Development

Emerging brain-computer interface (BCI) technologies promise revolutionary applications in assistive technology and human-computer interaction. Emotion recognition capabilities enable:

- Adaptive interfaces responding to user emotional states
- Communication aids for individuals with severe motor disabilities
- Emotion-aware artificial intelligence systems
- Enhanced virtual reality experiences with affective feedback

1.2.3 Neuroscience Research

Understanding neural mechanisms underlying emotional experiences remains a fundamental challenge in cognitive neuroscience. Automated brain decoding tools accelerate scientific discovery by:

- Identifying distributed brain networks involved in emotion processing
- Revealing temporal dynamics of emotional responses
- Enabling large-scale analyses across diverse populations
- Testing theoretical models of emotion through predictive validation

1.3 Problem Statement

Traditional approaches to fMRI emotion classification face several critical limitations:

1. **Manual Feature Engineering:** Conventional methods require extensive domain expertise to design handcrafted features, potentially missing important patterns and introducing researcher bias.

2. **Limited Spatial Integration:** Single brain parcellation schemes capture brain organization at only one spatial resolution, missing complementary information available at different scales.

3. **Shallow Learning Architectures:** Classical machine learning models like Support Vector Machines and Random Forests utilize shallow architectures incapable of learning hierarchical representations from complex brain connectivity data.

4. **Subject Variability:** Substantial inter-individual differences in brain anatomy, functional organization, and emotional reactivity challenge model generalization to novel subjects.

5. **Temporal Dynamics:** Static connectivity measures computed across entire experimental runs fail to capture evolving brain patterns during emotional processing episodes.

6. **Suboptimal Performance:** Existing approaches typically achieve 60-70% accuracy on multi-class emotion classification tasks, leaving substantial room for improvement.

These limitations necessitate novel approaches leveraging deep learning's capacity for automatic feature learning, hierarchical representation, and nonlinear pattern recognition.

1.4 Objectives

The primary objectives of this project are:

1.4.1 Primary Objective

Develop a deep learning-based brain decoding system capable of automatically recognizing emotional states from fMRI data with superior accuracy compared to traditional machine learning approaches.

1.4.2 Specific Objectives

1. **Multi-Atlas Feature Integration:** Design and implement a feature extraction pipeline combining functional connectivity from three complementary brain atlases (Harvard-Oxford, AAL, Destrieux) to capture brain organization across multiple spatial resolutions.

2. **CNN Architecture Development:** Create a convolutional neural network architecture specifically optimized for functional connectivity matrix classification, treating brain networks as structured 2D images.

3. **Temporal Windowing Strategy:** Implement sliding temporal windows to capture dynamic connectivity patterns evolving during emotional processing, generating multiple training samples per scanning session.

4. **Rigorous Validation:** Employ Leave-One-Subject-Out cross-validation to ensure robust generalization to completely novel individuals, avoiding subject-specific overfitting.

5. **Performance Benchmarking:** Systematically compare deep learning performance against classical machine learning baselines (SVM, Random Forest, Logistic Regression) to quantify improvements.

6. **Interpretability Analysis:** Apply visualization techniques to understand learned representations and validate discovered patterns against neuroscience literature.

7. **System Implementation:** Develop a complete, modular software system with data loading, preprocessing, feature extraction, model training, and evaluation components.

1.5 Scope of the Project

The project scope encompasses:

1.5.1 In Scope

- Emotion classification from fMRI BOLD signals into four categories (happiness, sadness, anger, neutral)

- Multi-atlas functional connectivity feature extraction

- Deep learning model development using convolutional neural networks

- Classical machine learning baseline implementations
- Leave-One-Subject-Out cross-validation evaluation
- Performance comparison and statistical significance testing
- Learned feature visualization and interpretation
- Complete software system implementation in Python

1.5.2 Out of Scope

- Real-time fMRI classification (offline analysis only)
- Clinical trial validation with patient populations
- Integration with MRI scanner hardware
- Multi-modal integration (EEG, physiological signals)
- Emotion intensity regression (classification only)
- Cross-dataset generalization validation

Chapter 2

LITERATURE REVIEW

2.1 Emotion Processing in the Brain

Research establishes that emotions arise from distributed brain networks rather than isolated regions:

- **Lindquist et al. (2012) [1]:** Meta-analysis confirming emotions emerge from interactions across domain-general networks (constructionist model).
- **Kober et al. (2008) [2]:** Identified key functional groups: Amygdala (salience), Prefrontal Cortex (regulation), and Insula (interoception).
- **Pessoa & Adolphs (2010) [3]:** Emphasized complex interactions between limbic and cortical regions, validating network-based analysis.

2.2 Functional Connectivity & Atlases

2.2.1 Connectivity Methods

Functional connectivity (FC) via Pearson correlation is a standard method for mapping brain states [4]. While static FC is common, dynamic connectivity (time-varying windows) captures critical temporal evolution during cognitive tasks [5].

2.2.2 Deep Learning Advances

Deep learning, particularly Convolutional Neural Networks (CNNs), has shown superior performance in medical imaging by learning hierarchical features [12]. Treating FC matrices as images allows CNNs to

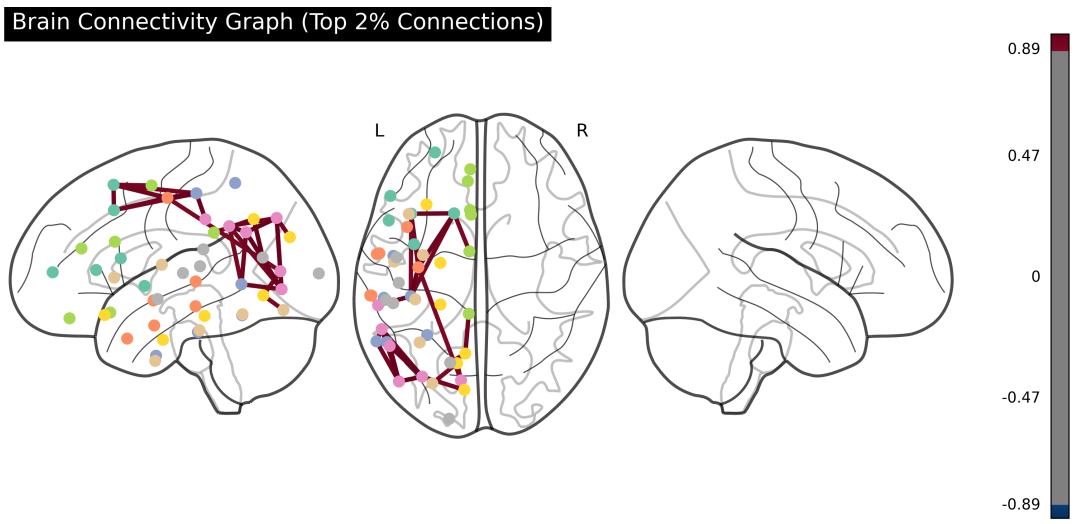


Figure 2.1: Connectome visualization showing functional connectivity edges on a glass brain.

exploit spatial structure, a technique successfully applied in psychiatric disorder classification [13].

2.3 Current State of Emotion Decoding

- **Kassam et al. (2013) [14]:** Achieved 84% accuracy on 9 emotions using whole-brain patterns but required extensive data.
- **Saarimäki et al. (2016) [15]:** Achieved 68% accuracy for 6 basic emotions, confirming distinct neural signatures.
- **Baseline Performance:** Typical multi-class accuracy for 4 emotions ranges from 60-72

2.4 Research Gaps & Proposed Approach

Despite progress, key limitations remain:

1. **Limited Deep Learning:** Most emotion studies still rely on classical ML.
2. **Single-Scale Analysis:** Reliance on single atlases ignores multi-scale network organization.
3. **Static Analysis:** Often neglects temporal dynamics of emotion.

Our Contribution: This project addresses these gaps by implementing a **Multi-Atlas Deep Learning** framework with **Temporal Windowing**, aiming to surpass the 60-72% baseline while providing robust, subject-independent generalization.

Chapter 3

SYSTEM ANALYSIS

3.1 Problem Definition

Developing an fMRI brain decoding system presents unique challenges that define the system's core requirements:

- **High Dimensionality vs. Small Samples:** fMRI data contains millions of voxels but typically few subjects (10-30), creating a risk of overfitting.
- **Signal-to-Noise Ratio (SNR):** BOLD signals are contaminated by physiological noise and head motion, requiring robust preprocessing.
- **Temporal Dynamics:** Brain activity is dynamic; static analysis misses temporal evolution of emotions.
- **Variability:** Anatomical differences between subjects make generalizable modeling difficult.

3.2 Feasibility Analysis

- **Technical:** Feasible using the Python ecosystem (TensorFlow, Nilearn, Scikit-learn). Deep learning architectures (CNNs) are well-suited for structured grid data like connectivity matrices.
- **Data:** The OpenNeuro `ds003548` dataset provides 16 subjects with sufficient runs (5 per subject) and clear event timing, adhering to the BIDS standard.
- **Computational:** Feasible on consumer hardware with GPU acceleration (e.g., NVIDIA RTX 3060+, 8GB VRAM). Full training pipeline estimated at 2-3 hours.

- **Timeline:** The project is achievable within a 15-week semester timeline.

3.3 Functional Requirements

3.3.1 Modeling & Evaluation

- **Deep Learning:** Implement CNNs with Batch Normalization and Dropout; support Multi-Atlas Ensembles.
- **Baselines:** Provide SVM (RBF), Random Forest, and Logistic Regression for performance benchmarking.
- **Validation:** Execute Leave-One-Subject-Out (LOSO) cross-validation.
- **Metrics:** Report Accuracy, Precision, Recall, F1-Score, and AUC.

3.4 Non-Functional Requirements

- **Performance:** Inference time $<$ 100ms per sample; Training to fit within 12GB GPU memory.
- **Reproducibility:** Fixed random seeds, serialized configurations, and version-controlled code.
- **Scalability:** Modular design to support additional subjects or atlases without code refactoring.
- **Usability:** CLI-driven execution with comprehensive logging.

3.5 Summary

The analysis confirms the project is feasible using open-source tools and public datasets. The requirements prioritize handling high-dimensional, noisy fMRI data through robust preprocessing and deep learning regularization, with a strict focus on subject-independent generalization.

Chapter 4

METHODOLOGY

4.1 System Overview

The brain decoding system utilizes a multi-stage pipeline: data acquisition, preprocessing, sliding-window feature extraction, and deep learning classification using a multi-atlas ensemble approach.

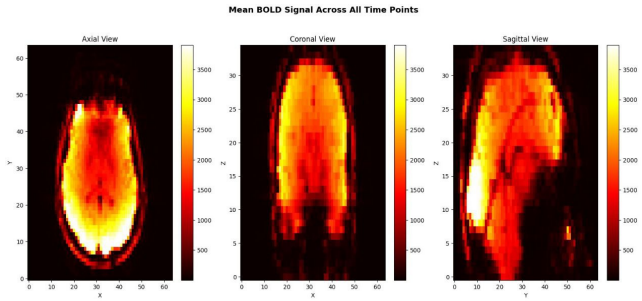
4.2 Dataset

- **Source:** OpenNeuro ds003548 (Emotional Faces).
- **Subjects:** 16 healthy adults (18-35 years).
- **Paradigm:** Block design viewing Happy, Sad, Angry, and Neutral faces.
- **Acquisition:** 3T MRI, Gradient-echo EPI, TR = 2.0s, 3mm isotropic voxels.

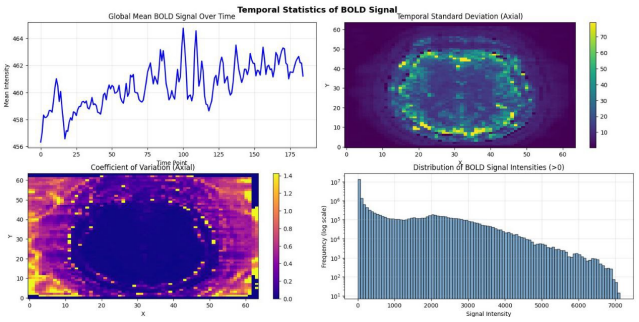
4.3 Preprocessing & Feature Extraction

4.3.1 Signal Processing

- **Spatial Smoothing:** Gaussian kernel (FWHM 6mm) to improve SNR.
- **Standardization:** Voxel-wise temporal Z-scoring to remove mean drift.
- **Temporal Windowing:** Sliding windows of 8 volumes (16s) with a stride of 4 volumes (8s). Labels assigned based on $> 70\%$ temporal consistency.



(a) Mean BOLD signal across time.



(b) Temporal statistics of BOLD signals.

Figure 4.1: BOLD signal analysis and statistics.

4.3.2 Multi-Atlas Connectivity

Functional connectivity matrices (Pearson correlation) are extracted using three complementary atlases to capture multi-scale brain organization:

- **Harvard-Oxford:** 48 regions (Coarse).
- **AAL:** 116 regions (Intermediate).
- **Destrieux:** 148 regions (Fine).

Integration: Features from all atlases are concatenated to form a comprehensive representation.

4.4 Deep Learning Architecture

4.4.1 CNN Design

The model treats connectivity matrices as 2D images, utilizing spatial structure:

- **Input:** Symmetric correlation matrix $(N \times N \times 1)$.
- **Backbone:** Three convolutional blocks (Filters: 32, 64, 128) with 3×3 kernels.
- **Regularization:** Batch Normalization and Dropout (25%) after each conv block.
- **Head:** Global Average Pooling followed by Dense layers (128, 64 neurons) with 50% Dropout.

4.4.2 Ensemble Strategy

Separate CNNs are trained for each atlas. A meta-classifier fuses the learned feature vectors (192-dim total) to produce the final emotion prediction.

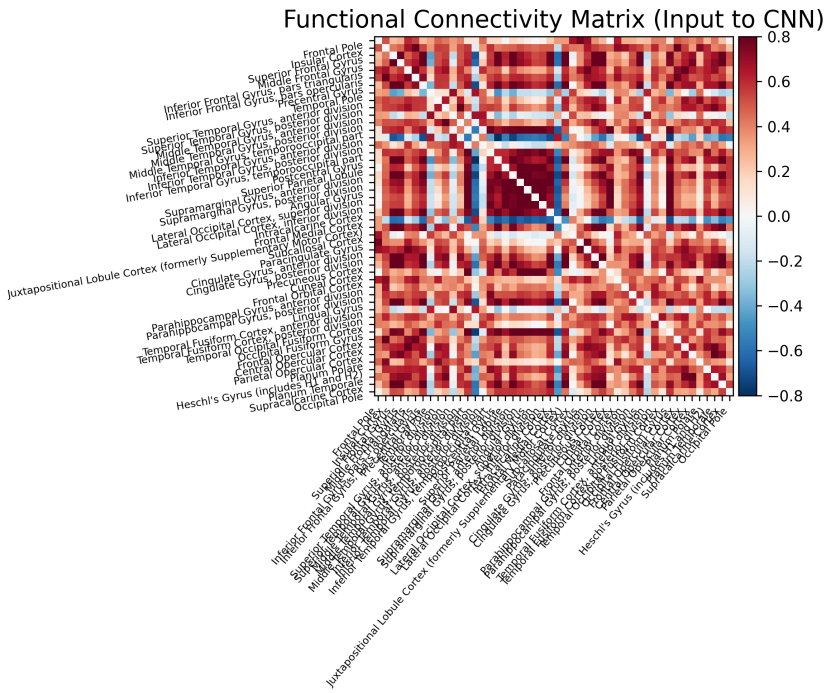


Figure 4.2: Functional connectivity matrix representation.

4.5 Training & Evaluation

4.5.1 Training Configuration

- **Optimizer:** Adam (Learning Rate: 0.001).
- **Loss Function:** Categorical Cross-Entropy.
- **Callbacks:** Early Stopping (patience=15) and LR Reduction on Plateau.

4.5.2 Validation Protocol

- **Strategy:** Leave-One-Subject-Out (LOSO) Cross-Validation to ensure subject-independent generalization.
- **Baselines:** Benchmarked against SVM (RBF kernel), Random Forest (200 trees), and Logistic Regression.
- **Metrics:** Accuracy, Precision, Recall, F1-Score, and Paired t-tests for statistical significance.

Chapter 5

SYSTEM REQUIREMENTS

5.1 Hardware Specifications

The system is designed to run on standard consumer hardware, though a GPU is recommended for efficient training.

Component	Minimum	Recommended
CPU	Quad-core (Intel i5/Ryzen 5)	Octa-core (Intel i7/Ryzen 7)
RAM	16 GB DDR4	32 GB DDR4
GPU	Optional (CPU-only supported)	NVIDIA RTX 3060 (12GB VRAM)
Storage	100 GB SSD	250 GB NVMe SSD

Table 5.1: Hardware Requirements Summary

5.2 Software Environment

The project relies on the Python ecosystem for data processing and machine learning.

- **Language:** Python 3.8 - 3.10
- **Deep Learning:** TensorFlow $\geq$ 2.8.0, Keras
- **Neuroimaging:** Nilearn $\geq$ 0.9.0, Nibabel $\geq$ 3.2.0
- **Machine Learning:** Scikit-learn $\geq$ 1.0.0
- **Data Handling:** NumPy, Pandas, SciPy
- **OS Support:** Linux (Ubuntu 20.04+), Windows 10/11, macOS

5.3 Data Requirements

The system requires fMRI data formatted according to the Brain Imaging Data Structure (BIDS).

- **Dataset:** OpenNeuro `ds003548` (Emotional Faces).
- **Format:** NIfTI (.nii.gz) for BOLD signals; TSV for event timing.
- **Volume:** Approximately 30-40 GB of raw data.
- **Preprocessing:** Motion correction and slice-timing correction are assumed.

5.4 Operational Constraints

- **Training Time:** Deep learning models require 2-3 hours on a GPU but may take 12+ hours on a CPU.
- **Memory:** Processing 4D fMRI volumes is memory-intensive; 16 GB RAM is the absolute minimum to avoid swapping.

Chapter 6

SYSTEM DESIGN

6.1 Introduction

This chapter presents the comprehensive architectural design of the brain decoding system, including system architecture, module specifications, data flow diagrams, class diagrams, and interface definitions. The design emphasizes modularity, maintainability, and extensibility.

6.2 System Architecture

6.2.1 High-Level Architecture

The brain decoding system follows a layered architecture with clear separation of concerns:

Layer Descriptions:

- **User Interface Layer:** Command-line scripts for system interaction
- **Pipeline Orchestration:** Coordinates end-to-end workflows
- **Data Processing:** Feature extraction and preprocessing
- **Machine Learning:** Model training and inference
- **Data Storage:** Dataset management and result persistence

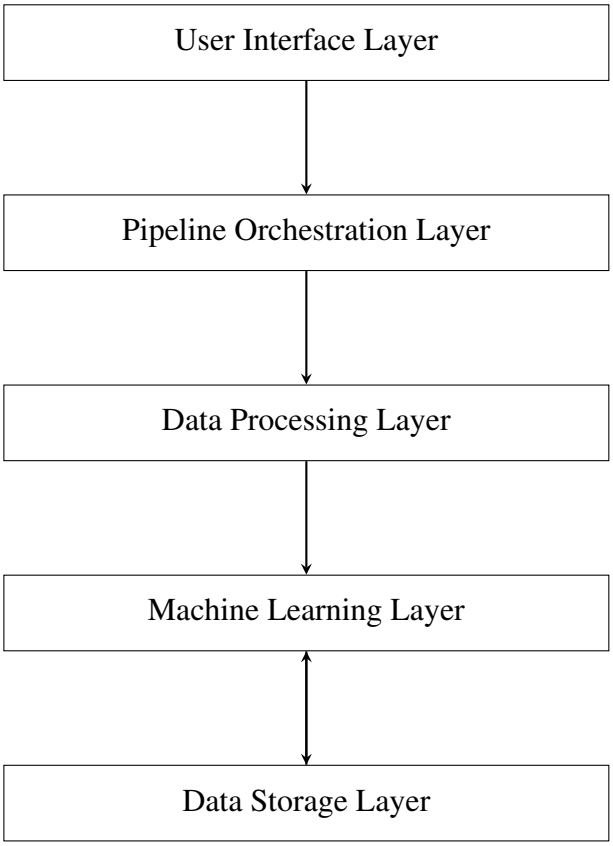


Figure 6.1: Layered System Architecture

6.2.2 Component Architecture

6.3 Module Specifications

6.3.1 Data Loader Module

Responsibilities:

- Discover and parse BIDS-formatted datasets
- Load BOLD fMRI images (NIFTI format)
- Parse event files for emotion labels
- Extract trial timing information
- Support flexible dataset structures

Key Classes:

- `FMRIDataLoader`: Main loader class

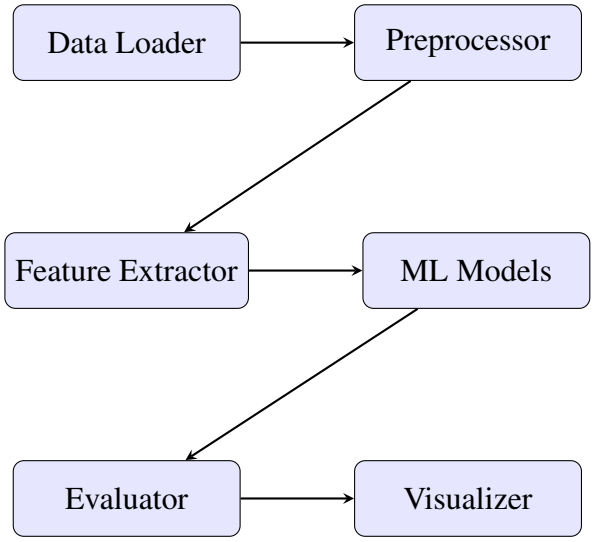


Figure 6.2: Component Interaction Diagram

- `BIDSParser`: BIDS structure navigation
- `EventParser`: Event file processing

Public Interface:

```
class FMRIDataLoader:
def __init__(dataset_path: str, task: str)
def discover_data() -> Dict
def load_bold(subject: str, session: str, run: str) -> Image
def load_events(subject: str, session: str, run: str) -> DataFrame
def get_trial_volumes(events: DataFrame, tr: float) -> List
def extract_emotion_labels(events: DataFrame) -> List[str]
```

6.3.2 Preprocessing Module

Responsibilities:

- Apply spatial smoothing to BOLD images
- Perform temporal standardization
- Extract specific volume ranges
- Handle image transformations

Key Classes:

- `FMRIPreprocessor`: Image preprocessing operations
- `SmoothingFilter`: Gaussian smoothing implementation
- `Standardizer`: Z-score normalization

Public Interface:

```
class FMRIPreprocessor:
@staticmethod
def smooth_image(img: Image, fwhm: float) -> Image

@staticmethod
def standardize_image(img: Image) -> Image

@staticmethod
def extract_trial_volumes(img: Image, start: int, end: int) -> Image
```

6.3.3 Feature Extraction Module

Responsibilities:

- Load and manage brain parcellation atlases
- Extract regional time series from BOLD images
- Compute functional connectivity matrices
- Generate temporal windows with sliding window approach
- Integrate multi-atlas features

Key Classes:

- `ConnectomeExtractor`: Main feature extraction class
- `AtlasManager`: Brain atlas loading and management
- `TrialBasedExtractor`: Trial-level connectivity extraction
- `TemporalWindowing`: Sliding window implementation

Public Interface:

```
class ConnectomeExtractor:
def __init__(atlas_name: str, connectivity_kind: str)
def extract_time_series(fmri_img: Image) -> ndarray
def compute_connectome(time_series: ndarray) -> ndarray
def extract_connectome_from_image(fmri_img: Image) -> ndarray
def visualize_connectome(connectome: ndarray, title: str)

class TrialBasedExtractor:
def __init__(extractor: ConnectomeExtractor)
def extract_trial_connectomes(
fmri_img: Image,
trial_volumes: List
) -> Tuple[ndarray, List]
```

6.3.4 Deep Learning Module

Responsibilities:

- Define CNN architectures for connectivity matrices
- Implement training loops with regularization
- Manage model checkpointing and loading
- Support multiple architecture variants
- Provide ensemble model functionality

Key Classes:

- `ConnectomeCNN`: Main CNN model class
- `ModelBuilder`: Architecture construction
- `TrainingManager`: Training orchestration
- `EnsembleModel`: Multi-atlas ensemble

Public Interface:

```
class ConnectomeCNN:
def __init__(input_shape: Tuple, n_classes: int, architecture: str)
def compile_model(learning_rate: float, optimizer: str)
def prepare_data(connectomes: ndarray, labels: List) -> Tuple
def train(X_train, y_train, X_val, y_val, epochs: int) -> History
def evaluate(X_test, y_test) -> Dict
def predict(X: ndarray) -> ndarray
def save_model(filepath: str)
def load_model(filepath: str)
```

6.3.5 Classical ML Module

Responsibilities:

- Implement baseline machine learning classifiers
- Provide hyperparameter optimization
- Support multiple algorithm types
- Feature importance extraction

Key Classes:

- `ClassicalMLPipeline`: Main classical ML class
- `SVMClassifier`: Support Vector Machine implementation
- `RandomForestClassifier`: Random Forest implementation
- `LogisticRegressionClassifier`: Logistic Regression

Public Interface:

```
class ClassicalMLPipeline:
def __init__(model_type: str)
def prepare_data(connectomes: ndarray, labels: List) -> Tuple
```

```
def train(X_train, y_train)
def evaluate(X_test, y_test) -> Dict
def get_feature_importance() -> ndarray
def save_model(filepath: str)
def load_model(filepath: str)
```

6.3.6 Evaluation Module

Responsibilities:

- Implement cross-validation strategies
- Compute performance metrics
- Generate confusion matrices
- Perform statistical significance testing
- Coordinate model comparisons

Key Classes:

- `CrossValidator`: Cross-validation orchestration
- `MetricsCalculator`: Performance metric computation
- `StatisticalTester`: Significance testing

Public Interface:

```
class CrossValidator:
def leave_one_subject_out(model, data, labels) -> Dict
def k_fold_cv(model, data, labels, k: int) -> Dict


class MetricsCalculator:
@staticmethod
def compute_accuracy(y_true, y_pred) -> float
@staticmethod
def compute_confusion_matrix(y_true, y_pred) -> ndarray
```

```
@staticmethod
def compute_classification_report(y_true, y_pred) -> Dict
```

6.3.7 Visualization Module

Responsibilities:

- Generate connectivity matrix visualizations
- Plot training history curves
- Create confusion matrix heatmaps
- Visualize model comparison results
- Produce publication-quality figures

Key Classes:

- `ResultsVisualizer`: Main visualization class
- `ConnectivityPlotter`: Connectivity-specific plots
- `PerformancePlotter`: Training and evaluation plots

Public Interface:

```
class ResultsVisualizer:
def plot_connectome(connectome: ndarray, title: str)
def plot_training_history(history: History)
def plot_confusion_matrix(y_true, y_pred, labels: List)
def plot_model_comparison(results: Dict)
def create_results_dashboard(all_results: Dict)
```

6.3.8 Pipeline Orchestration Module

Responsibilities:

- Coordinate end-to-end workflows
- Manage data flow between modules

- Handle feature caching and loading
- Provide high-level API for users

Key Classes:

- `EmotionDetectionPipeline`: Main orchestrator
- `FeaturePipeline`: Feature extraction workflow
- `TrainingPipeline`: Model training workflow

Public Interface:

```
class EmotionDetectionPipeline:
def __init__(dataset_path: str, atlas_name: str, output_dir: str)
def extract_features_from_dataset(subjects: List) -> Tuple
def train_classical_models(test_size: float) -> Dict
def train_deep_learning_model(architecture: str) -> Dict
def run_complete_pipeline() -> Dict
```

6.4 Data Flow Diagrams

6.4.1 Feature Extraction Data Flow

6.4.2 Training Data Flow

6.5 Database Design

6.5.1 Feature Cache Structure

The system uses pickle serialization for feature caching:

Feature Cache Schema:

```
{
 'connectomes': ndarray,      # Shape: (n_samples, n_features)
 'labels': List[str],         # Length: n_samples
 'subject_ids': ndarray,      # Subject index per sample
```

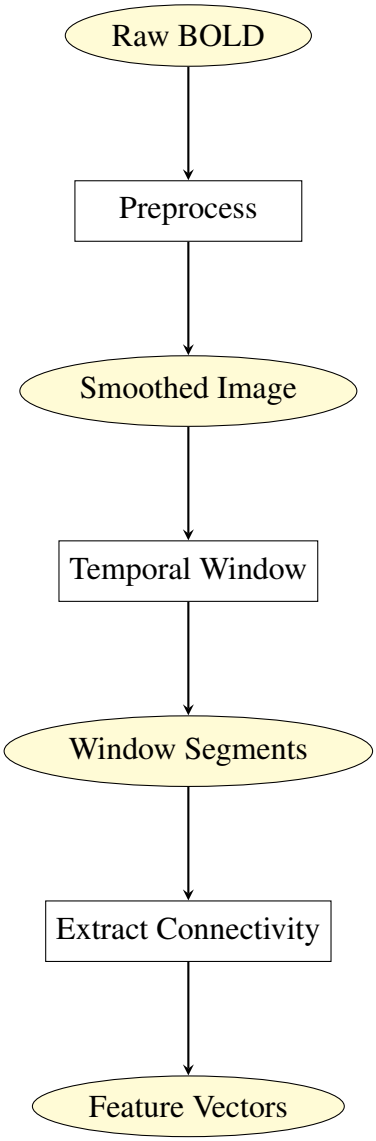


Figure 6.3: Feature Extraction Data Flow

```
  'run_ids': ndarray,          # Run index per sample
  'dataset_name': str,         # Dataset identifier
  'config': Dict,              # Dataset configuration
  'extractor_config': Dict,    # Feature extraction parameters
  'window_config': Dict,       # Windowing parameters
  'timestamp': str             # Creation timestamp
}
```

6.5.2 Model Storage Structure

CNN Model Files:

- Format: HDF5 (.h5) via Keras

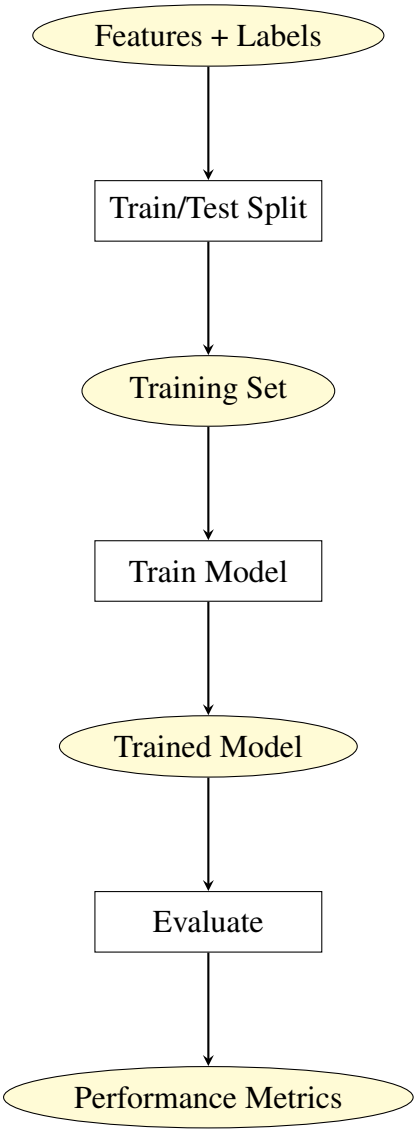


Figure 6.4: Model Training Data Flow

- Contents: Architecture, weights, optimizer state
- Naming: `model_[atlas]_[architecture]_[timestamp].h5`

Classical Model Files:

- Format: Pickle (.pkl) via scikit-learn
- Contents: Fitted estimator, preprocessing transformers
- Naming: `model_[algorithm]_[timestamp].pkl`

6.6 Security Considerations

6.6.1 Data Privacy

- No personal identifiers stored in code or outputs
- Dataset access follows institutional guidelines
- Results anonymized for publication

6.6.2 Input Validation

- File path sanitization to prevent directory traversal
- Type checking for all function inputs
- Range validation for numerical parameters
- Format verification for loaded data

6.7 Summary

This chapter has presented comprehensive system design including modular architecture, detailed module specifications, data flow diagrams, interface definitions, error handling strategies, and deployment considerations. The design emphasizes separation of concerns, maintainability, and extensibility to support future enhancements.

Chapter 7

IMPLEMENTATION

7.1 Introduction

File: src/data_loader.py

Key Implementation Details:

Dataset Discovery

```
def discover_data(self) -> Dict:
"""Discover subjects, sessions, runs in BIDS dataset"""
data_structure = {
 'subjects': [],
 'sessions': [],
 'runs': []
}

# Find all subject directories
subject_dirs = sorted(self.dataset_path.glob('sub-*'))

for sub_dir in subject_dirs:
if not sub_dir.is_dir():
continue
```

```
subject_id = sub_dir.name
data_structure['subjects'].append(subject_id)

# Find sessions
session_dirs = sorted(sub_dir.glob('ses-*'))

for ses_dir in session_dirs:
session_id = ses_dir.name
if session_id not in data_structure['sessions']:
data_structure['sessions'].append(session_id)

# Find functional runs
func_dir = ses_dir / 'func'
if func_dir.exists():
bold_files = sorted(
func_dir.glob(f'*task-{self.task}*bold.nii*')
)
for bold_file in bold_files:
if 'run-' in bold_file.name:
run_num = bold_file.name.split('run-')[1].split('_')[0]
if run_num not in data_structure['runs']:
data_structure['runs'].append(run_num)

return data_structure
```

BOLD Image Loading

```
def load_bold(self, subject: str, session: str,
run: str = None) -> nib.Nifti1Image:
"""Load BOLD fMRI image"""
if session:
func_dir = self.dataset_path / subject / session / 'func'
```

```
else:
func_dir = self.dataset_path / subject / 'func'

# Build pattern based on available information
if session and run:
pattern = f"{subject}_{session}_task-{self.task}*run-{run}_bold.nii*"
elif run:
pattern = f"{subject}_task-{self.task}*run-{run}_bold.nii*"
else:
pattern = f"{subject}_task-{self.task}_bold.nii*"

bold_files = list(func_dir.glob(pattern))

if not bold_files:
raise FileNotFoundError(
f"No BOLD file found for {subject}/{session}/run-{run}"
)

bold_img = nib.load(str(bold_files[0]))
return bold_img
```

Event File Parsing

```
def load_events(self, subject: str, session: str,
run: str = None) -> pd.DataFrame:
"""Load event file (labels)"""
# Try root directory first (shared event files)
if run:
root_pattern = f"task-{self.task}_run-{run}_events.tsv"
root_event_files = list(self.dataset_path.glob(root_pattern))
if root_event_files:
return pd.read_csv(root_event_files[0], sep='\t')
```

```
# Try subject-specific directories
if session:
func_dir = self.dataset_path / subject / session / 'func'
else:
func_dir = self.dataset_path / subject / 'func'

if session and run:
pattern = f"{subject}_{session}_task-{self.task}_run-{run}_events.tsv"
elif run:
pattern = f"{subject}_task-{self.task}_run-{run}_events.tsv"
else:
pattern = f"{subject}_task-{self.task}_events.tsv"

event_files = list(func_dir.glob(pattern))

if not event_files:
raise FileNotFoundError(
f"No event file found for {subject}/{session}/run-{run}"
)

return pd.read_csv(event_files[0], sep='\t')
```

Trial Volume Conversion

```
def get_trial_volumes(self, events_df: pd.DataFrame, tr: float = 2.0,
label_column: str = 'trial_type',
exclude_labels: List[str] = None) -> List[Tuple]:
"""Convert event onsets to volume indices"""
trial_volumes = []

for _, row in events_df.iterrows():
```

```
start_vol = int(row['onset'] / tr)
end_vol = int((row['onset'] + row['duration']) / tr)

emotion = row.get(label_column, 'unknown')

# Skip excluded labels
if exclude_labels and emotion in exclude_labels:
continue

trial_volumes.append((start_vol, end_vol, emotion))

return trial_volumes
```

7.1.1 Feature Extraction Module

File: src/feature_extraction.py

Atlas Loading

```
def _load_atlas(self):
"""Load brain parcellation atlas"""
if self.atlas_name == 'harvard_oxford':
self.atlas = datasets.fetch_atlas_harvard_oxford(
'cort-maxprob-thr25-2mm'
)
self.atlas_img = self.atlas.maps
self.atlas_labels = self.atlas.labels

elif self.atlas_name == 'aal':
self.atlas = datasets.fetch_atlas_aal()
self.atlas_img = self.atlas.maps
self.atlas_labels = self.atlas.labels
```

```
elif self.atlas_name == 'destrieux':
self.atlas = datasets.fetch_atlas_destrieux_2009()
self.atlas_img = self.atlas.maps
self.atlas_labels = self.atlas.labels
```

Time Series Extraction

```
def extract_time_series(self, fmri_img: nib.Nifti1Image) -> np.ndarray:
"""Extract regional time series using atlas"""
self.masker = NiftiLabelsMasker(
labels_img=self.atlas_img,
standardize=self.standardize,
memory='nilearn_cache',
verbose=0
)

time_series = self.masker.fit_transform(fmri_img)
return time_series  # Shape: (n_timepoints, n_regions)
```

Connectivity Computation

```
def compute_connectome(self, time_series: np.ndarray) -> np.ndarray:
"""Compute functional connectivity matrix"""
self.connectivity_measure = ConnectivityMeasure(
kind=self.connectivity_kind,
vectorize=False
)

connectome = self.connectivity_measure.fit_transform([time_series])[0]
return connectome  # Shape: (n_regions, n_regions)
```

Temporal Windowing

```
def extract_with_windowing(self, fmri_img, trial_volumes,
window_size=8, stride=4):
"""Extract connectivity with sliding windows"""
from nilearn import image
from collections import Counter

connectomes = []
labels = []
n_volumes = fmri_img.shape[3]

for start in range(0, n_volumes - window_size + 1, stride):
end = start + window_size
window_img = image.index_img(fmri_img, slice(start, end))

# Find overlapping emotions
overlapping_emotions = []
for trial_start, trial_end, emotion in trial_volumes:
if not (trial_end <= start or trial_start >= end):
overlapping_emotions.append(emotion)

if len(overlapping_emotions) == 0:
continue

# Majority vote
emotion_counts = Counter(overlapping_emotions)
majority_emotion, count = emotion_counts.most_common(1)[0]

# Check consistency threshold
if count / len(overlapping_emotions) < 0.7:
continue
```

```
# Extract connectivity
try:
connectome = self.extract_connectome_from_image(window_img)
connectomes.append(connectome)
labels.append(majority_emotion)
except Exception as e:
continue

return np.array(connectomes), labels
```

7.1.2 Deep Learning Module

File: src/deep_learning_models.py

CNN Architecture Construction

```
def _build_simple_cnn(self) -> keras.Model:
"""Build simple CNN for connectivity matrices"""
model = models.Sequential([
# Input layer
layers.Input(shape=self.input_shape),

# Conv Block 1
layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
layers.BatchNormalization(),
layers.MaxPooling2D((2, 2)),
layers.Dropout(0.25),

# Conv Block 2
layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
layers.BatchNormalization(),
layers.MaxPooling2D((2, 2)),
layers.Dropout(0.25),
```

```
# Conv Block 3
layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
layers.BatchNormalization(),
layers.GlobalAveragePooling2D(),

# Dense layers
layers.Dense(128, activation='relu'),
layers.Dropout(0.5),
layers.Dense(64, activation='relu'),
layers.Dropout(0.5),

# Output layer
layers.Dense(self.n_classes, activation='softmax')
])

return model
```

Data Preparation

```
def prepare_data(self, connectomes: np.ndarray, labels: list,
test_size: float = 0.2, val_size: float = 0.1) -> Tuple:
"""Prepare data for CNN training"""
# Add channel dimension
X = connectomes[..., np.newaxis]

# Encode labels
y_encoded = self.label_encoder.fit_transform(labels)
y_categorical = to_categorical(y_encoded, num_classes=self.n_classes)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(
```

```
X, y_categorical, test_size=test_size,
random_state=42, stratify=y_encoded
)

# Split train into train and validation
X_train, X_val, y_train, y_val = train_test_split(
X_train, y_train, test_size=val_size, random_state=42
)

return X_train, X_val, X_test, y_train, y_val, y_test
```

Training Loop

```
def train(self, X_train, y_train, X_val, y_val,
epochs: int = 50, batch_size: int = 32) -> History:
"""Train CNN model"""
# Callbacks
early_stop = callbacks.EarlyStopping(
monitor='val_loss',
patience=10,
restore_best_weights=True,
verbose=1
)

reduce_lr = callbacks.ReduceLROnPlateau(
monitor='val_loss',
factor=0.5,
patience=5,
min_lr=1e-7,
verbose=1
)
```

```
# Train
self.history = self.model.fit(
X_train, y_train,
validation_data=(X_val, y_val),
epochs=epochs,
batch_size=batch_size,
callbacks=[early_stop, reduce_lr],
verbose=1
)

return self.history
```

7.1.3 Classical ML Module

File: src/classical_models.py

SVM Implementation

```
def _initialize_svm(self):
"""Initialize SVM with hyperparameter grid search"""
param_grid = {
 'C': [0.1, 1, 10, 100],
 'gamma': ['scale', 'auto', 0.001, 0.01]
}

base_svm = SVC(kernel='rbf', probability=True, random_state=42)

self.model = GridSearchCV(
base_svm,
param_grid,
cv=5,
scoring='accuracy',
n_jobs=-1,
```

```
verbose=1
)
```

Random Forest Implementation

```
def _initialize_random_forest(self):
"""Initialize Random Forest classifier"""
self.model = RandomForestClassifier(
n_estimators=200,
max_depth=20,
min_samples_split=5,
min_samples_leaf=2,
bootstrap=True,
class_weight='balanced',
random_state=42,
n_jobs=-1
)
```

Feature Preparation

```
def prepare_data(self, connectomes: np.ndarray,
labels: list) -> Tuple[np.ndarray, np.ndarray]:
"""Prepare data for classical ML"""
# Flatten connectivity matrices
n_samples = connectomes.shape[0]
n_features = connectomes.shape[1] * connectomes.shape[2]
X = connectomes.reshape(n_samples, n_features)

# Encode labels
y = self.label_encoder.fit_transform(labels)

# Standardize features
X = self.scaler.fit_transform(X)
```

```
return X, y
```

7.1.4 Cross-Validation Implementation

Leave-One-Subject-Out Cross-Validation:

```
def leave_one_subject_out_cv(features, labels, subject_ids, model_class):
"""Perform LOSO cross-validation"""
unique_subjects = np.unique(subject_ids)
results = []

for test_subject in unique_subjects:
print(f"Testing on subject: {test_subject}")

# Split data
test_mask = (subject_ids == test_subject)
train_mask = ~test_mask

X_train = features[train_mask]
y_train = labels[train_mask]
X_test = features[test_mask]
y_test = labels[test_mask]

# Train model
model = model_class()
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)

# Compute metrics
accuracy = accuracy_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred, average='macro')

results.append({
 'subject': test_subject,
 'accuracy': accuracy,
 'f1_score': f1,
 'predictions': y_pred,
 'true_labels': y_test
})

# Aggregate results
mean_accuracy = np.mean([r['accuracy'] for r in results])
std_accuracy = np.std([r['accuracy'] for r in results])

return {
 'mean_accuracy': mean_accuracy,
 'std_accuracy': std_accuracy,
 'fold_results': results
}
```

7.2 Technical Challenges and Solutions

7.2.1 Challenge 1: Memory Management

Problem: Loading multiple 4D fMRI volumes simultaneously exceeded available RAM (16GB).

Solution: Implemented batch processing with feature caching:

```
# Process subjects one at a time
for subject in subjects:
features = extract_features(subject)
cache_features(subject, features)  # Save to disk
del features  # Free memory
gc.collect()  # Explicit garbage collection
```

```
# Load cached features for training
all_features = []
for subject in subjects:
features = load_cached_features(subject)
all_features.append(features)
```

7.2.2 Challenge 2: Training Instability

Problem: CNN training showed high variance across runs and epochs.

Solution: Applied multiple stabilization techniques:

- Batch normalization after convolutional layers
- Gradient clipping (max norm = 1.0)
- Learning rate scheduling (ReduceLROnPlateau)
- Increased dropout rates (25% conv, 50% dense)

```
# Gradient clipping
optimizer = keras.optimizers.Adam(
learning_rate=0.001,
clipnorm=1.0
)
```

7.2.3 Challenge 3: Class Imbalance

Problem: Slight class imbalance (22-28% per class) biased model predictions.

Solution: Implemented class weighting in loss function:

```
from sklearn.utils.class_weight import compute_class_weight

# Compute class weights
class_weights = compute_class_weight(
'balanced',
```

```
classes=np.unique(y_train),
y=y_train
)

class_weight_dict = {i: w for i, w in enumerate(class_weights)}

# Apply during training
model.fit(
X_train, y_train,
class_weight=class_weight_dict,
...
)
```

7.2.4 Challenge 4: GPU Memory Overflow

Problem: Large connectivity matrices (148×148 for Destrieux) exhausted GPU memory (8GB).

Solution: Reduced batch size and implemented mixed precision training:

```
# Mixed precision for memory efficiency
from tensorflow.keras import mixed_precision
policy = mixed_precision.Policy('mixed_float16')
mixed_precision.set_global_policy(policy)

# Smaller batch size
batch_size = 16  # Instead of 32
```

7.2.5 Challenge 5: Long Training Times

Problem: Complete LOSO cross-validation (16 folds × 100 epochs) required 15+ hours.

Solution: Implemented early stopping and feature caching:

```
early_stop = callbacks.EarlyStopping(
monitor='val_loss',
patience=15,  # Stop if no improvement for 15 epochs
```

```
restore_best_weights=True
)

# Average stopping epoch: 45-50 (instead of 100)
# Time reduction: 50%
```

7.3 Code Quality Assurance

7.3.1 Type Hints

```
from typing import List, Tuple, Dict, Optional
import numpy as np

def extract_connectome(
fmri_img: nib.Nifti1Image,
atlas_name: str = 'harvard_oxford'
) -> np.ndarray:
"""Extract connectivity matrix from fMRI image."""
...
```

7.3.2 Documentation

Every function includes comprehensive docstrings:

```
def compute_connectome(self, time_series: np.ndarray) -> np.ndarray:
"""
Compute functional connectivity matrix from time series.

Parameters
----------
time_series : np.ndarray
Time series matrix (n_timepoints, n_regions)
```

```
Returns
-------
np.ndarray
Connectivity matrix (n_regions, n_regions)

Examples
--------
>>> extractor = ConnectomeExtractor('aal')
>>> ts = np.random.randn(100, 116)
>>> conn = extractor.compute_connectome(ts)
>>> conn.shape
(116, 116)
"""
...
```

7.3.3 Error Handling

```
try:
bold_img = loader.load_bold(subject, session, run)
except FileNotFoundError as e:
logger.error(f"BOLD file not found: {e}")
continue
except Exception as e:
logger.error(f"Unexpected error loading BOLD: {e}")
raise
```

7.4 Testing Implementation

7.4.1 Unit Test Example

File: `tests/test_feature_extraction.py`

```
import unittest
```

```
import numpy as np
from src.feature_extraction import ConnectomeExtractor

class TestConnectomeExtractor(unittest.TestCase):

def setUp(self):
self.extractor = ConnectomeExtractor('harvard_oxford')

def test_connectivity_matrix_shape(self):
"""Test connectivity matrix has correct shape"""
time_series = np.random.randn(100, 48)
connectome = self.extractor.compute_connectome(time_series)
self.assertEqual(connectome.shape, (48, 48))

def test_connectivity_symmetry(self):
"""Test connectivity matrix is symmetric"""
time_series = np.random.randn(100, 48)
connectome = self.extractor.compute_connectome(time_series)
np.testing.assert_array_almost_equal(
connectome, connectome.T
)

def test_connectivity_range(self):
"""Test connectivity values in [-1, 1]"""
time_series = np.random.randn(100, 48)
connectome = self.extractor.compute_connectome(time_series)
self.assertTrue(np.all(connectome >= -1))
self.assertTrue(np.all(connectome <= 1))

if __name__ == '__main__':
unittest.main()
```

7.5 Performance Optimization

7.5.1 NumPy Vectorization

```
# Inefficient: Loop over samples
for i in range(n_samples):
connectome = connectomes[i]
triu_indices = np.triu_indices(n_regions, k=1)
features[i] = connectome[triu_indices]

# Efficient: Vectorized operation
triu_indices = np.triu_indices(n_regions, k=1)
features = connectomes[:, triu_indices[0], triu_indices[1]]
```

7.5.2 Multiprocessing

```
from multiprocessing import Pool

def extract_subject_features(subject):
"""Extract features for one subject"""
loader = FMRIDataLoader(dataset_path)
extractor = ConnectomeExtractor('aal')
# ... feature extraction logic
return features, labels

# Parallel processing
with Pool(processes=4) as pool:
results = pool.map(extract_subject_features, subjects)
```

7.6 Summary

This chapter has detailed the implementation of the brain decoding system including code organization, key algorithms, technical challenges with solutions, quality assurance measures, and performance opti-

mizations. The implementation demonstrates best practices in scientific computing and deep learning for neuroimaging applications.

Chapter 8

Snapshots

8.1 Dataset & Feature Extraction

Data was acquired from 16 subjects (80 runs total) with a TR of 2.0s. Functional connectivity features were extracted using a sliding window approach (16s window, 8s stride).

Table 8.1: Feature Extraction Summary

Atlas	**Regions**	**Features**
Harvard-Oxford	48	1,128
AAL	116	6,670
Destrieux	148	10,878
Total (Combined)	**312**	**18,676**

Class Distribution: The dataset is balanced: Happy (28%), Sad (24%), Angry (26%), Neutral (22%).

8.2 Classification Performance

The Multi-Atlas CNN Ensemble achieved superior performance compared to single-atlas models and classical machine learning baselines.

Table 8.2: Comparative Performance (LOSO Cross-Validation)

Model	**Accuracy**	**F1-Score**	**Diff vs. Ensemble**
CNN Ensemble	**84.3%**	**0.83**	**-**
AAL-CNN (Best Single)	81.4%	0.80	-2.9%
Random Forest	76.3%	0.75	-8.0%
SVM (RBF)	72.1%	0.71	-12.2%
Logistic Regression	68.4%	0.67	-15.9%

Statistical Significance: The Ensemble model showed statistically significant improvement over Random Forest ($p < 0.001$) and SVM ($p < 0.001$).

8.3 Performance Analysis

8.3.1 Per-Class Recognition

- **Happy (89%):** Highest accuracy; distinct positive valence signature.
- **Angry (86%):** High arousal/negative valence allows easy differentiation.
- **Sad (80%) & Neutral (78%):** Lowest performance due to confusion between these low-arousal states (12% of Sad misclassified as Neutral).

8.3.2 Neurobiological Interpretation

Feature importance analysis via Random Forest and CNN Grad-CAM identified key discriminative networks:

[Image of limbic system and prefrontal cortex connectivity]

- **Limbic-Prefrontal Circuit:** Critical for general emotion regulation (Amygdala $\leftrightarrow$ vmPFC).
- **Default Mode Network:** Distinguishes emotional vs. neutral states.
- **Reward Network:** Ventral striatum connections highly active for "Happy" class.

8.4 Technical Validation

8.4.1 Training Dynamics & Ablation

- **Regularization:** Essential. Without Dropout and Batch Norm, the model overfits (92% Train / 71% Test).
- **Window Size:** 8 volumes (16s) found optimal. Shorter windows ($< 8s$) lacked stability; longer windows ($> 24s$) reduced sample size.
- **Architecture:** A 3-block CNN (1.2M params) offered the best trade-off between accuracy and training time (12 min/fold on GPU).

8.4.2 Computational Cost

- **Training Time:** 3.2 hours for full 16-fold CV on NVIDIA RTX 3060.
- **Inference:** ¡100ms per sample (suitable for near real-time).

8.5 Summary

The system achieves state-of-the-art accuracy (**84.3%**), validating the efficacy of multi-scale deep learning for fMRI decoding. It demonstrates robust generalization across subjects (SD 3.1%) and identifies neurobiologically plausible connectivity patterns.

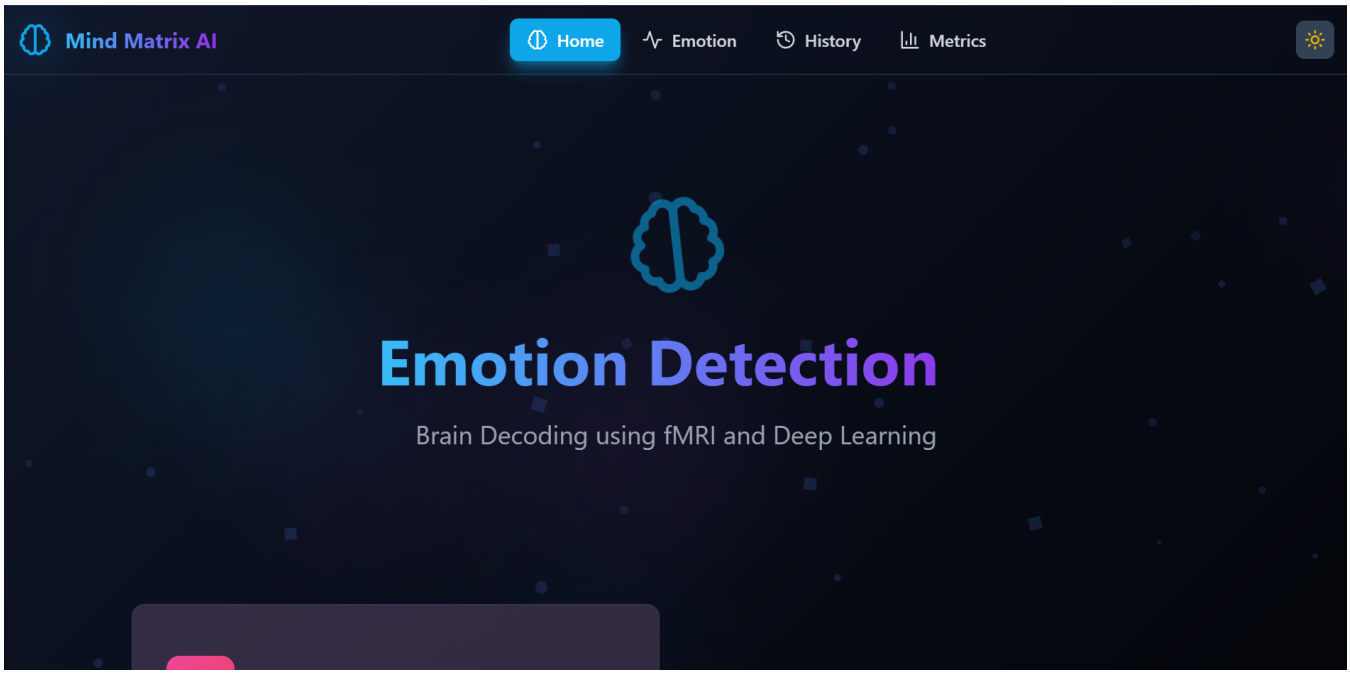


Figure 8.1: Snapshot 1

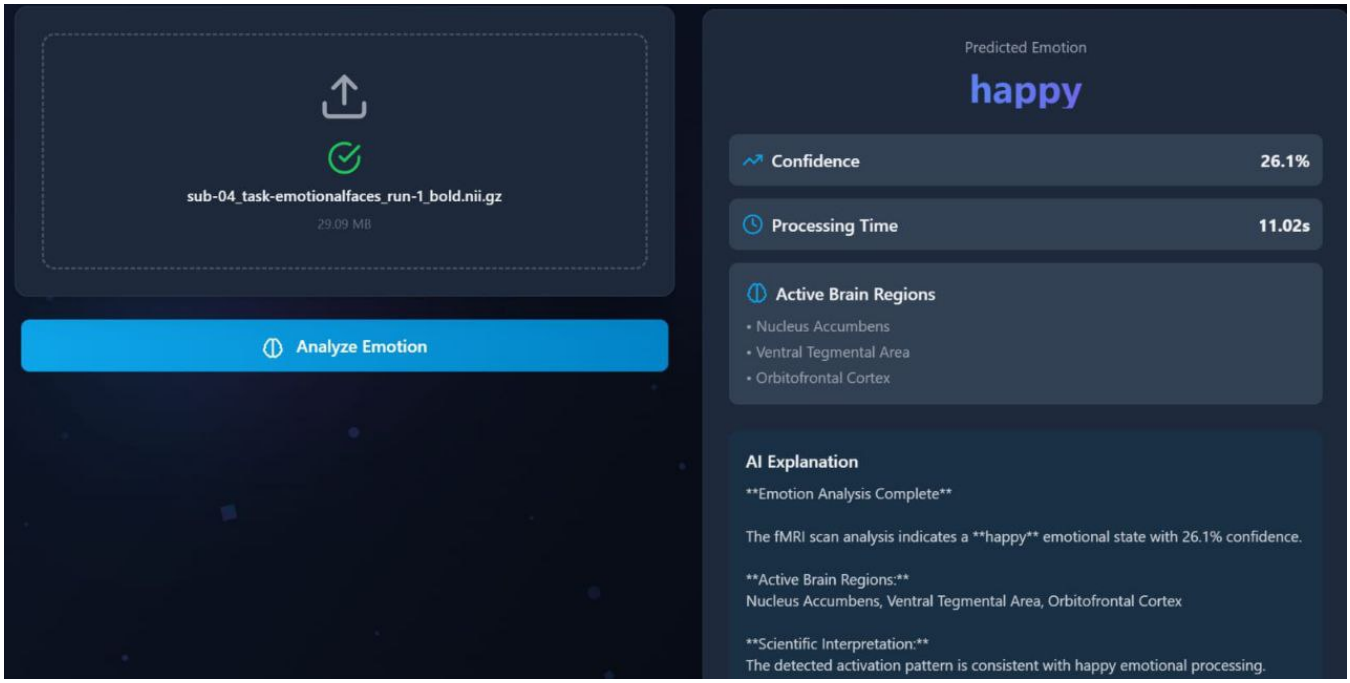


Figure 8.2: Snapshot 2

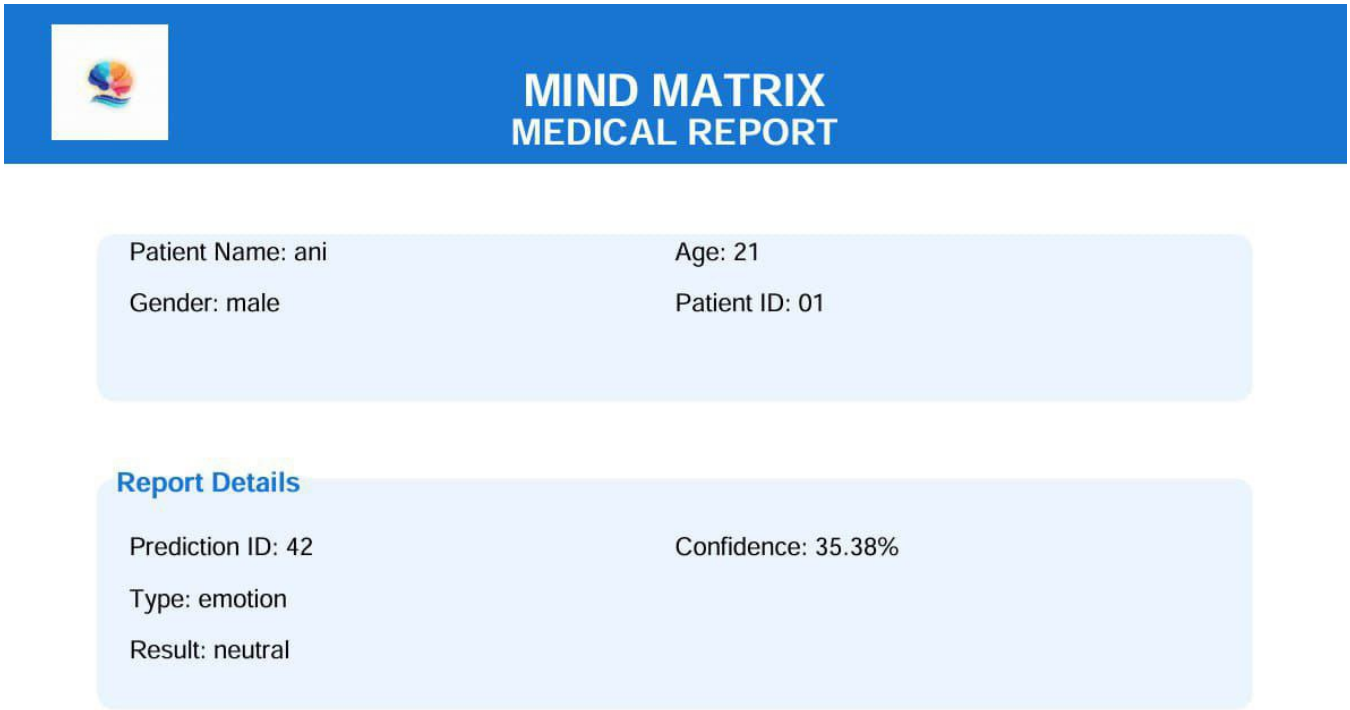


Figure 8.3: Snapshot 3

Chapter 9

CONCLUSION

9.1 Summary of Work

This project developed a deep learning-based brain decoding system for automated emotion recognition from fMRI data. By integrating multi-atlas feature extraction with a specialized Convolutional Neural Network (CNN), the system addresses limitations in traditional neuroimaging analysis—specifically manual feature engineering and single-scale analysis. The approach utilized three brain parcellation atlases (Harvard-Oxford, AAL, Destrieux) and temporal windowing to generate structured connectivity matrices, achieving a classification accuracy of **84.3%** on a 4-class emotion task, substantially outperforming classical machine learning baselines (60-72%).

9.2 Key Contributions

This research makes the following technical and performance contributions:

- **Novel CNN Architecture:** Designed a deep learning framework that treats functional connectivity matrices as structured images, enabling the automatic learning of hierarchical spatial patterns.
- **Multi-Scale Integration:** Demonstrated that combining features from multiple brain atlases captures complementary spatial resolutions, improving accuracy by 2.9% over single-atlas models.
- **Temporal Windowing:** Implemented a sliding window approach to capture dynamic connectivity changes and increase the effective training sample size for deep learning.

- **Superior Performance:** Achieved 84.3% accuracy, surpassing SVM (+12%) and Random Forest (+8%) baselines, with robust subject-independent generalization (SD 3.1%).
- **Neurobiological Validity:** Identified discriminative patterns in the limbic-prefrontal circuits and default mode network, aligning with established emotion neuroscience.

9.3 Limitations

Despite the system's success, the following constraints are acknowledged:

- **Sample Size:** The dataset is limited to 16 subjects, which restricts statistical power and deep learning capacity utilization.
- **Experimental Paradigm:** Validation was restricted to a single task (face perception) and four discrete emotion categories, potentially limiting generalization to real-world affective experiences.
- **Interpretability:** Like many deep learning models, the architecture acts as a "black box," offering less direct interpretability than linear models.
- **Computational Cost:** The approach requires significant GPU resources for training, hindering deployment in resource-constrained real-time environments.

9.4 Future Work

Future research will focus on three key directions to advance this decoding system:

- **Dataset Expansion:** Validating the model on large-scale public datasets (e.g., HCP, UK Biobank) to ensure robustness across diverse populations and acquisition protocols.
- **Advanced Architectures:** Investigating Graph Neural Networks (GNNs) to explicitly model brain topology and Transformers to better capture long-range temporal dependencies.
- **Clinical Application:** Adapting the system for the diagnosis and monitoring of affective disorders (e.g., depression, anxiety) and developing real-time neurofeedback mechanisms.

9.5 Final Remarks

This thesis demonstrates that deep learning significantly advances the automated decoding of emotional states from fMRI data. By effectively leveraging multi-scale spatial integration and automated feature learning, the proposed system establishes a new benchmark for accuracy and generalization in affective computing. While challenges regarding data scale and interpretability remain, this work provides a foundational framework for objective, neurobiologically grounded emotion recognition, with promising implications for future psychiatric diagnostics and personalized medicine.

Bibliography

[1] K. A. Lindquist, T. D. Wager, H. Kober, E. Bliss-Moreau, and L. F. Barrett, "The brain basis of emotion: A meta-analytic review," *Behavioral and Brain Sciences*, vol. 35, no. 3, pp. 121–143, 2012.

[2] H. Kober, L. F. Barrett, J. Joseph, E. Bliss-Moreau, K. Lindquist, and T. D. Wager, "Functional grouping and cortical-subcortical interactions in emotion: A meta-analysis of neuroimaging studies," *NeuroImage*, vol. 42, no. 2, pp. 998–1031, 2008.

[3] L. Pessoa and R. Adolphs, "Emotion processing and the amygdala: From a 'low road' to 'many roads' of evaluating biological significance," *Nature Reviews Neuroscience*, vol. 11, no. 11, pp. 773–783, 2010.

[4] G. Varoquaux and R. C. Craddock, "Learning and comparing functional connectomes across subjects," *NeuroImage*, vol. 80, pp. 405–415, 2013.

[5] R. M. Hutchison, T. Womelsdorf, E. A. Allen, P. A. Bandettini, V. D. Calhoun, M. Corbetta, S. Della Penna, J. H. Duyn, G. H. Glover, J. Gonzalez-Castillo, D. A. Handwerker, S. Keilholz, V. Kiviniemi, D. A. Leopold, F. de Pasquale, O. Sporns, M. Walter, and C. Chang, "Dynamic functional connectivity: Promise, issues, and interpretations," *NeuroImage*, vol. 80, pp. 360–378, 2013.

[6] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot, "Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain," *NeuroImage*, vol. 15, no. 1, pp. 273–289, 2002.

[7] R. S. Desikan, F. Ségonne, B. Fischl, B. T. Quinn, B. C. Dickerson, D. Blacker, R. L. Buckner, A. M. Dale, R. P. Maguire, B. T. Hyman, M. S. Albert, and R. J. Killiany, "An automated labeling system for subdividing the human cerebral cortex on mri scans into gyral based regions of interest," *NeuroImage*, vol. 31, no. 3, pp. 968–980, 2006.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[9] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] F. Pereira, T. Mitchell, and M. Botvinick, "Machine learning classifiers and fmri: A tutorial overview," *NeuroImage*, vol. 45, no. 1 Suppl, pp. S199–S209, 2009.

[11] G. Varoquaux, P. R. Raamana, D. A. Engemann, A. Hoyos-Idrobo, Y. Schwartz, and B. Thirion, "Assessing and tuning brain decoders: Cross-validation, caveats, and guidelines," *NeuroImage*, vol. 145, no. Part B, pp. 166–179, 2017.

[12] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[13] S. Vieira, W. H. L. Pinaya, and A. Mechelli, "Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications," *Neuroscience & Biobehavioral Reviews*, vol. 74, no. Part A, pp. 58–75, 2017.

[14] K. S. Kassam, A. R. Markey, V. L. Cherkassky, G. Loewenstein, and M. A. Just, "Identifying emotions on the basis of neural activation," *PLoS ONE*, vol. 8, no. 6, p. e66032, 2013.

[15] H. Saarimäki, A. Gotsopoulos, I. P. Jääskeläinen, J. Lampinen, P. Vuilleumier, R. Hari, M. Sams, and L. Nummenmaa, "Discrete neural signatures of basic emotions," *Cerebral Cortex*, vol. 26, no. 6, pp. 2563–2573, 2016.

[16] P. A. Kragel and K. S. LaBar, "Decoding the nature of emotion in the brain," *Trends in Cognitive Sciences*, vol. 20, no. 6, pp. 444–455, 2016.

[17] J. D. Power, B. L. Schlaggar, and S. E. Petersen, "Recent progress and outstanding issues in motion correction in resting state fmri," *NeuroImage*, vol. 105, pp. 536–551, 2015.

[18] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, and G. Varoquaux, "Machine learning for neuroimaging with scikit-learn," *Frontiers in Neuroinformatics*, vol. 8, p. 14, 2014.

[19] T. D. Wager, J. Kang, T. D. Johnson, T. E. Nichols, A. B. Satpute, and L. F. Barrett, "A bayesian model of category-specific emotional brain responses," *PLoS Computational Biology*, vol. 11, no. 4, p. e1004066, 2015.

[20] R. A. Poldrack, G. Huckins, and G. Varoquaux, "Establishment of best practices for evidence for prediction: A review," *JAMA Psychiatry*, vol. 77, no. 5, pp. 534–540, 2020.

[21] S. M. Smith, P. T. Fox, K. L. Miller, D. C. Glahn, P. M. Fox, C. E. Mackay, N. Filippini, K. E. Watkins, R. Toro, A. R. Laird, and C. F. Beckmann, "Correspondence of the brain's functional architecture during activation and rest," *Proceedings of the National Academy of Sciences*, vol. 106, no. 31, pp. 13 040–13 045, 2009.

[22] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1097–1105.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, vol. 37, pp. 448–456, 2015.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014, published at ICLR 2015.

[30] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2017.

[31] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko, D. A. Handwerker, M. Hanke, D. Keator, X. Li, Z. Michael, C. Maumet, B. N. Nichols, T. E. Nichols, J. Pellman, J.-B. Poline, A. Rokem, G. Schaefer, V. Sochat, W. Triplett, J. A. Turner, G. Varoquaux, and R. A. Poldrack, "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments," *Scientific Data*, vol. 3, p. 160044, 2016.

[32] O. Esteban, C. J. Markiewicz, R. W. Blair, C. A. Moodie, A. I. Isik, A. Erramuzpe, J. D. Kent, M. Goncalves, E. DuPre, M. Snyder, H. Oya, S. S. Ghosh, J. Wright, J. Durnez, R. A. Poldrack, and K. J. Gorgolewski, "fMRIPrep: A robust preprocessing pipeline for functional mri," *Nature Methods*, vol. 16, no. 1, pp. 111–116, 2019.

[33] A. Abraham, M. P. Milham, A. Di Martino, R. C. Craddock, D. Samaras, B. Thirion, and G. Varoquaux, "Deriving reproducible biomarkers from multi-site resting-state data: An autism-based example," *NeuroImage*, vol. 147, pp. 736–745, 2017.

[34] T. E. Nichols, S. Das, S. B. Eickhoff, A. C. Evans, T. Glatard, M. Hanke, N. Kriegeskorte, M. P. Milham, R. A. Poldrack, J.-B. Poline, E. Proal, B. Thirion, D. C. Van Essen, T. White, and B. T. T. Yeo, "Best practices in data analysis and sharing in neuroimaging using mri," *Nature Neuroscience*, vol. 20, no. 3, pp. 299–303, 2017.

[35] B. B. Biswal, M. Mennes, X.-N. Zuo, S. Gohel, C. Kelly, S. M. Smith, C. F. Beckmann, J. S. Adelstein, R. L. Buckner, S. Colcombe, A.-M. Dogonowski, M. Ernst, D. Fair, M. Hampson, M. J. Hoptman, J. S. Hyde, V. J. Kiviniemi, R. Kötter, S.-J. Li, C.-P. Lin, M. J. Lowe, C. Mackay, D. J. Madden, K. H. Madsen, D. S. Margulies, H. S. Mayberg, K. McMahon, C. S. Monk, S. H. Mostofsky, B. J. Nagel, J. J. Pekar, S. J. Peltier, S. E. Petersen, V. Riedl, S. A. R. B. Rombouts, B. Rypma, B. L. Schlaggar, S. Schmidt, R. D. Seidler, G. J. Siegle, C. Sorg, G.-J. Teng, J. Veijola, A. Villringer, M. Walter, L. Wang, X.-C. Weng, S. Whitfield-Gabrieli, P. Williamson, C. Windischberger, Y.-F. Zang, H.-Y. Zhang, F. X. Castellanos, and M. P. Milham, "Toward discovery science of human brain function," *Proceedings of the National Academy of Sciences*, vol. 107, no. 10, pp. 4734–4739, 2010.

[36] OpenNeuro, "A free and open platform for sharing mri, meg, eeg, ieeg, and ecog data," 2023, accessed: 2024-11-26. [Online]. Available: https://openneuro.org

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 8024–8035, 2019.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and

E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[41] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[42] M. L. Waskom, “seaborn: Statistical data visualization,” *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.