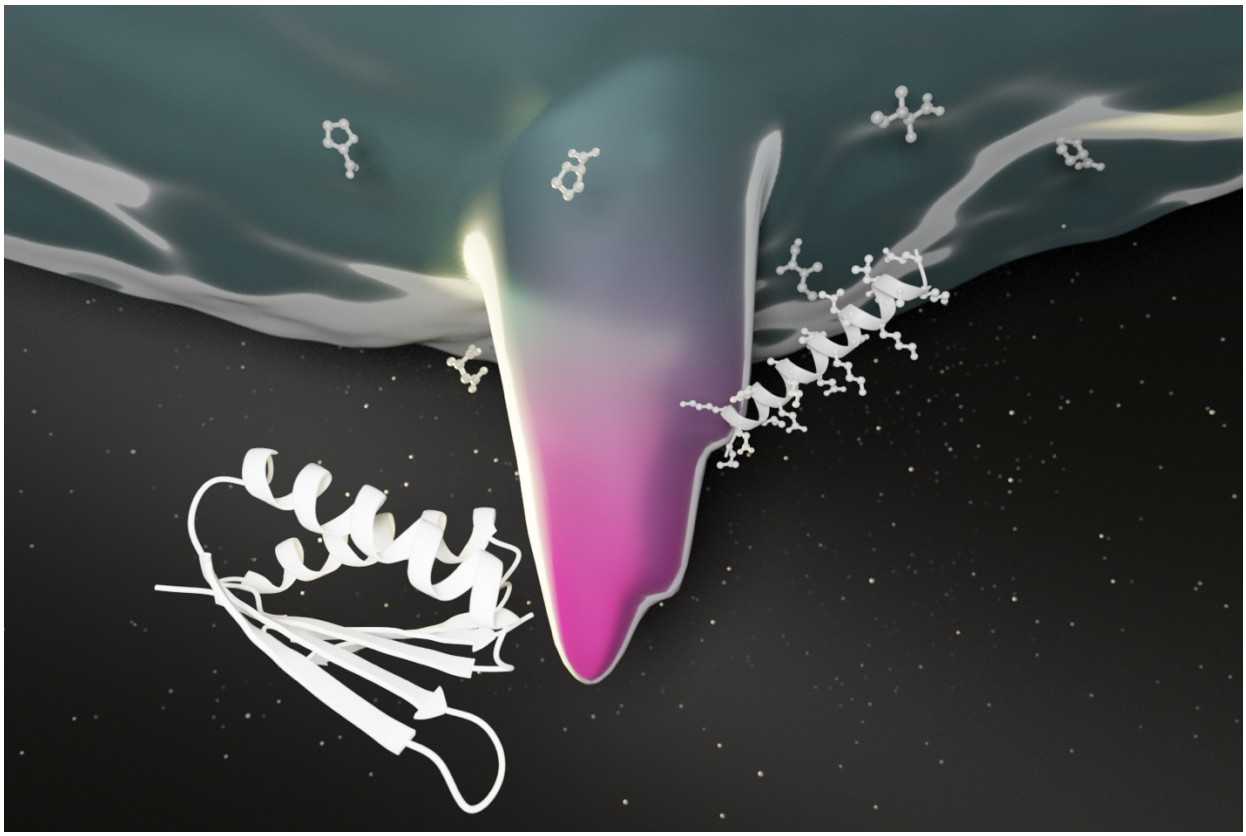

# DEEP-POCKET

## SBI/PYT PROJECT


**Authors**:

**Allal El Hammad**:
allal.elhommad01@estudiant.upf.edu
**Javier Herranz**:
javier.herranz01@estudiant.upf.edu
**Daniel Perez**
daniel.perez15@estudiant.upf.edu

## Index

# Deep-Pocket Documentation

## Package Overview

Deep Pocket is a Python package designed to predict the binding pocket of a protein structure provided by the user. Leveraging deep learning techniques, the package utilizes a neural network model trained on a dataset comprising 1000 Protein Data Bank (PDB) files. With its focus on accuracy and efficiency, Deep Pocket aims to assist bioinformaticians in identifying binding pockets within protein structures, aiding in drug discovery, protein function analysis, and molecular docking studies.

Deep Pocket is primarily aimed at bioinformaticians, computational biologists, and researchers working in the field of structural biology. It provides a valuable tool for analyzing protein structures and identifying potential binding sites, facilitating drug discovery, protein-protein interaction studies, and other molecular biology applications.

Whether you are conducting research in academia, working in the pharmaceutical industry, or exploring protein structure-function relationships, Deep Pocket offers a robust and efficient solution for predicting binding pockets with high accuracy and reliability.

**Main Features:**
- **Predict Binding Pockets:** Utilize the trained neural network model to predict the binding pockets of a protein structure.
- **Bioinformatic Applications:** Targeted towards bioinformaticians and researchers in the life sciences for various computational biology tasks.
- **Deep Learning Techniques:** Harnesses the power of deep learning to accurately identify binding sites within protein structures.
- **Ease of Use:** Designed with user-friendliness in mind, allowing for seamless integration into existing workflows.

## Installation

To use Deep Pocket, it is required to have a working version of the DSSP software installed on your system. DSSP is used for extracting secondary structure and solvent accessibility information from protein structures.
Installing DSSP:
**For macOS users, DSSP can be installed using Homebrew:**

```
brew install dssp
```

**For Linux users, DSSP can be installed using apt:**

```
sudo apt install dssp
```

DSSP can also be installed using Conda. However, please note that this may not be compatible with computers using Apple Silicon (M1, M2, etc.) chips:
conda install salilab::dssp

Make sure to have DSSP properly installed and accessible in your system before using Deep Pocket. The recommended version of DSSP is 4.0.4, but any newer version that includes the mkdssp binary should work seamlessly with Deep Pocket.

# Option 1

**To install Deep Pocket, please follow these steps:**

**1. Clone the Repository**
Clone the Deep Pocket repository from GitHub to your local machine:

```
git clone https://github.com/aelhammad/deep-pocket
```

**2. Navigate to the Repository**
Navigate to the cloned Deep Pocket repository directory:

```
cd deep_pocket
```

**3. Create a Virtual Environment (Optional but Recommended)**
It is recommended to create a virtual environment to manage dependencies for Deep Pocket. If you prefer to use virtual environments, you can create one using the following command:

```
python3 -m venv venv
```

Activate the virtual environment:
**-On macOS/Linux:**

```
source venv/bin/activate
```

**-On Windows:**

```
venv\Scripts\activate
```

**4. Install Requirements**
Install the required dependencies listed in the requirements.txt file using pip:
pip install -r requirements.txt
This command will install the necessary packages, including:

- Biopython
- NumPy
- Torch
- Torchvision
- Torchaudio
- Pandas
- Scikit-learn
- matplotlib

# Option 2

To install Deep Pocket, please follow these steps:

**1. Clone the Repository**

Clone the Deep Pocket repository from GitHub to your local machine:

```
git clone https://github.com/aelhammad/deep-pocket
```

**2. Navigate to the Repository**
Navigate to the cloned Deep Pocket repository directory:

```
cd deep_pocket
```

**3. Create a Virtual Environment (Optional but Recommended)**
It is recommended to create a virtual environment to manage dependencies for Deep Pocket. If you prefer to use virtual environments, you can create one using the following command:

```
python3 -m venv venv
```

Activate the virtual environment:
**-On macOS/Linux:**

```
source venv/bin/activate
```

**-On Windows:**

```
venv\Scripts\activate
```

**Step 4: Install the Package**
Install the Deep Pocket package and its dependencies using setup.py:

```
python setup.py install
```

***Verify Installation**
To verify that Deep Pocket and its dependencies were installed successfully, you can run the following command:

```
python -c "import deep_pocket; print('Deep Pocket installed successfully')"
```

If installed correctly, you should see the message "Deep Pocket installed successfully" printed in the terminal.
or run in python:

```
import deep_pocket
```

# Documentation

## model.py

### Function preprocess_features

```
def preprocess_features(features):
```

### Description

Preprocess the features for training.

**Args:**

**features (dict)**: Dictionary containing features.

**Returns:**

dict: Dictionary containing preprocessed features.

## Class MyModel

```
class MyModel(nn.Module):
```

### Description

Neural network model for binary classification.

### methods

```
def forward(self, x):
```

Forward pass of the neural network.

# process_data.py

## Function find_matched_pdb_files

```
def find_matched_pdb_files(pdb_dir, pocket_dir):
```

### Description

Find matched PDB files and their corresponding pocket files.

**Args:**

**pdb_dir (str):** Path to the directory containing PDB files.

**pocket_dir (str):** Path to the directory containing pocket PDB files.

**Returns**:

**tuple**: A tuple containing lists of matched PDB and pocket file paths.

**Example usage:**

```
pdb_dir = '/path/to/pdb/directory'
pocket_dir = '/path/to/pocket/directory'
matched_pdb_files, matched_pocket_files = find_matched_pdb_files(pdb_dir,
pocket_dir)
print("Matched PDB files:", matched_pdb_files)
print("Matched pocket files:", matched_pocket_files)
```

## Function split_data

```
def split_data(matched_files, test_size=0.2, val_size=0.25):
```

### Description

Split data into training, validation, and test sets.

**Args**:

**data (list)**: A list of data to be split.
**train_ratio (float, optional)**: Ratio of training data. Defaults to 0.8.
**val_ratio (float, optional)**: Ratio of validation data. Defaults to 0.1.
**test_ratio (float, optional)**: Ratio of test data. Defaults to 0.1.

**Returns:**

**tuple**: A tuple containing lists of training, validation, and test data.

**Example usage:**

```
matched_files = list(zip(matched_pdb_files, matched_pocket_files))  # Assuming matched_pdb_files and matched_pocket_files are already obtained
train_files, val_files, test_files = split_data(matched_files)
print("Training files:", train_files)
print("Validation files:", val_files)
print("Test files:", test_files)
```

## Function save_features_pickle

```
def save_features_pickle(data, filepath):
```

### Description

Save features dictionary as a pickle file.

**Args**:

**features_dict (dict):** Dictionary containing features.
**filename (str):** Name of the pickle file to save.

**Example usage:**

```
data_to_save = {'example': 'data'}
save_features_pickle(data_to_save, 'example_data.pkl')
```

## Function load_features_pickle

```
def load_features_pickle(filepath):
```

### Description

Process PDB and pocket PDB files, extract features, and save them as pickle files.

**Args:**

**pdb_dir (str)**: Path to the directory containing PDB files.

**pocket_dir (str)**: Path to the directory containing pocket PDB files.

**Example usage**

```
loaded_data = load_features_pickle('example_data.pkl')
print("Loaded data:", loaded_data)
```

### Function process_data

```
def process_data(pdb_dir, pocket_dir):
```

#### Description

Process PDB and pocket PDB files, extract features, and save them as pickle files.

**Args**:

pdb_dir (str): Path to the directory containing PDB files.

pocket_dir (str): Path to the directory containing pocket PDB files.

**Example usage**

```
pdb_dir = '/path/to/pdb/directory'
pocket_dir = '/path/to/pocket/directory'
process_data(pdb_dir, pocket_dir)
```

## feature_extraction.py

### Class ProteinFeatures

```
class ProteinFeatures:
```

#### Description

Class used to extract features from a protein structure.

**Attributes:**

**pdb_file (str):** Path to the PDB file containing the protein structure.

**pocket_pdb_file (str, optional):** Path to the pocket PDB file.

**Example usage**

```
pdb_file = '/path/to/pdb/file.pdb'
pocket_pdb_file = '/path/to/pocket/pdb/file.pdb'
pf = ProteinFeatures(pdb_file, pocket_pdb_file)
features = pf.extract_features()
print(features)
```

#### methods

```
def extract_sequence(self):
```

Description

Extract the amino acid sequence from the PDB file.

**Returns**:

**dict:** A dictionary containing the frequencies of each amino acid in the sequence.

**Example usage**

```
pf = ProteinFeatures(pdb_file)
sequence = pf.extract_sequence()
print(sequence)
```

```
def calculate_total_contact(self):
```

Description

Calculate the total contact for each residue based on a distance threshold.

**Returns:**

**dict:** A dictionary containing the total contact for each residue.

**Example usage**

```
pf = ProteinFeatures(pdb_file)
total_contact = pf.calculate_total_contact()
print(total_contact)
# Example usage
pf = ProteinFeatures(pdb_file)
total_contact = pf.calculate_total_contact()
print(total_contact)
```

```
def extract_secondary_structure(self):
```

Description

Extract secondary structure and solvent accessibility using DSSP.

**Returns**:

**dict**: A dictionary containing secondary structure and solvent accessibility information for each residue.

**Example usage**

```
pf = ProteinFeatures(pdb_file)
secondary_structure = pf.extract_secondary_structure()
print(secondary_structure)
```

```
def load_pocket_residues(self):
```

Description

Load the pocket residues from a pocket PDB file.

**Returns:**

**set**: A set containing tuples of chain and residue IDs representing pocket residues.

**Example usage**

```
pf = ProteinFeatures(pdb_file, pocket_pdb_file)
pocket_residues = pf.load_pocket_residues()
print(pocket_residues)
```

```
def one_hot_encoding(self):
```

Description

Create one-hot encoding for amino acids and secondary structure.

**Returns:**

**tuple:** A tuple containing dictionaries of one-hot encodings for amino acids and secondary structure.

**Example usage**

```
pf = ProteinFeatures(pdb_file)
encoding_ss, encoding_aa = pf.one_hot_encoding()
print(encoding_ss)
print(encoding_aa)
```

```
def extract_features(self):
```

Description

Extract features from the protein structure.
Features to extract:

- PDB ID
- Residue Name
- In Pocket
- Secondary Structure
- Solvent Accessibility
- Psi Angle
- Phi Angle
- Total Contact

**Returns:**

**list**: A list of dictionaries containing features for each residue.

**Example usage**

```
pf = ProteinFeatures(pdb_file, pocket_pdb_file)
features = pf.extract_features()
print(features)
```

## Function download_pdb_files

```
def download_pdb_files(pocket_pdb_directory, pdb_directory):
```

Description

Download PDB files from the given pocket PDB directory and save them in the specified PDB directory.

**Parameters:**

**- pocket_pdb_directory (str):** Path to the directory containing the pocket PDB files.

**- pdb_directory (str):** Path to the directory where the downloaded PDB files will be saved.

**Example usage**

```
pocket_pdb_directory = '/path/to/pocket/pdb/directory'
pdb_directory = '/path/to/pdb/directory'
downloaded_pdb_files = download_pdb_files(pocket_pdb_directory,
pdb_directory)
print("Downloaded PDB files:", downloaded_pdb_files)
```

## Function featurizer

```
def featurizer(test_matched_files):
```

Description

Extract features from a list of PDB files and return a list of dictionaries.
To train the model you need to process (process_data.py) the data before using the featurizer.
To process the data: find_matched_pdb_files -> split_data -> save_features_pickle
Features calculated:

'PDB_ID'
'Residue_Name'
'In_Pocket'
'Secondary_structure'
'Solvent_accessibility'
'Psi_angle'
'Phi_angle'
'Total_contact'

**Args**:

**test_matched_files (list):** A list of tuples containing paths to PDB files and their corresponding pocket PDB files.

**Returns**:

**list**: A list of dictionaries containing features for each residue.

**Example usage**

```
test_matched_files = [
('/path/to/pdb/file1.pdb', '/path/to/pocket/pdb/file1_pocket.pdb'),
('/path/to/pdb/file2.pdb', '/path/to/pocket/pdb/file2_pocket.pdb')
 ]
features = featurizer(test_matched_files)
print("Extracted features:", features)
```

## predict.py

### Function predict_pdb

```
def predict_pdb(pdb_file_path, verbose=False):
```

#### Description

Predict binding pocket residues in a given PDB file using a pre-trained model.

**Parameters:**

**pdb_file_path (str):** Path to the input PDB file.

**verbose (bool):** Flag to enable verbose mode.

**Example usage:**

```
python3 predict.py ../example_pdb/pdb.pdb
```

# Troubleshooting and Support

If you encounter any issues or have any questions about Deep Pocket, please feel free to contact us:


- **Allal El Hammad**: allal.elhommad01@estudiant.upf.edu
- **Javier Herranz**: javier.herranz01@estudiant.upf.edu
- **Daniel Perez**: daniel.perez15@estudiant.upf.edu


# License and Legal Information