Run Immediately

Libraries

```
# Normal Lib
import os
import pandas as pd
import numpy as np
import pickle
import math
import tensorflow as tf
import random

# keras
from keras import Model
import keras
from keras.layers import Conv3D, Input, MaxPooling3D, BatchNormalization, Dense, Dropout, Flatten, AveragePooling

from keras.optimizers import Adam, RMSprop, SGD
from keras.regularizers import L2

# Metrics
from scipy.stats import pearsonr, spearmanr # Pearson R best
from keras.metrics import AUC, MeanAbsoluteError, Precision, Recall, Accuracy
from sklearn.metrics import matthews_corrcoef, mean_squared_error, r2_score
from keras.activations import linear
# from sklearn.metrics import mean_squared_error,

# rms = mean_squared_error(y_actual, y_predicted, squared=False)
# Import File
from zipfile import ZipFile
from google.colab import drive
import csv


!pip install rdkit
from rdkit import Chem

# Import File
from zipfile import ZipFile
from google.colab import drive

drive.mount('/content/gdrive/')
dataset_folder = '/content/gdrive/MyDrive/Final'
files = os.listdir(dataset_folder)
for file in files:
  if '.zip' in file:
    file_path = os.path.join(dataset_folder, file)
    with ZipFile(file_path, 'r') as f:
      f.extractall()
```

```
Collecting rdkit
  Downloading rdkit-2023.3.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (29.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 29.7/29.7 MB 31.6 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from rdkit) (1.22.4)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from rdkit) (8.4.0)
Installing collected packages: rdkit
Successfully installed rdkit-2023.3.2
Mounted at /content/gdrive/
```

Classes

```
#Converts the protein-ligand complexes into 4D tensor.
class Feature_extractor():
    def __init__(self):
        self.atom_codes = {}
        #'others' includs metal atoms and B atom. There are no B atoms on training and test sets.

        others = ([3,4,5,11,12,13]+list(range(19,32))+list(range(37,51))+list(range(55,84)))

        # C and N atoms can be hybridized in three ways and S atom can be hybridized in two ways here.
        # Hydrogen atom is also considered for feature extraction.

        atom_types = [1,(6,1),(6,2),(6,3),(7,1),(7,2),(7,3),8,15,(16,2),(16,3),34,[9,17,35,53],others]

        for i, j in enumerate(atom_types):
            if type(j) is list:
                for k in j:
                    self.atom_codes[k] = i

            else:
                self.atom_codes[j] = i
        self.sum_atom_types = len(atom_types)

    #Onehot encoding of each atom. The atoms in protein or ligand are treated separately.
    def encode(self, atomic_num, molprotein):
        encoding = np.zeros(self.sum_atom_types*2)
        if molprotein == 1:
            encoding[self.atom_codes[atomic_num]] = 1.0
        else:
            encoding[self.sum_atom_types+self.atom_codes[atomic_num]] = 1.0

        return encoding

    #Get atom coords and atom features from the complexes.
    def get_features(self, molecule, molprotein):
        coords = []
        features = []

        # molecule = Chem.MolFromPDBFile(protein_test_path, False, False, 1)
        molecule_conf = molecule.GetConformer()
        molecule_positions = molecule_conf.GetPositions()

        possible_hybridization_list = [
        Chem.rdchem.HybridizationType.UNSPECIFIED,
        Chem.rdchem.HybridizationType.S,
        Chem.rdchem.HybridizationType.SP,
        Chem.rdchem.HybridizationType.SP2,
        Chem.rdchem.HybridizationType.SP3,
        Chem.rdchem.HybridizationType.SP3D,
        Chem.rdchem.HybridizationType.SP3D2
        ]
        for idx, pos in enumerate(molecule_positions):
          coords.append(pos)
          atom = molecule.GetAtomWithIdx(int(idx))
          # print("A")
          # print(atom.GetHybridization())
          if atom.GetAtomicNum() in [6,7,16]:

            hyb = possible_hybridization_list.index(atom.GetHybridization())
            if hyb < 1:
              hyb = 2
            atomicnum = (atom.GetAtomicNum(), hyb)
            features.append(self.encode(atomicnum,molprotein))
          else:
            features.append(self.encode(atom.GetAtomicNum(),molprotein))

        coords = np.array(coords, dtype=np.float32)
        features = np.array(features, dtype=np.float32)

        return coords, features

    #Define the rotation matrixs of 3D stuctures.
    def rotation_matrix(self, t, roller):
        if roller==0:
            return np.array([[1,0,0],[0,np.cos(t),np.sin(t)],[0,-np.sin(t),np.cos(t)]])
        elif roller==1:
            return np.array([[np.cos(t),0,-np.sin(t)],[0,1,0],[np.sin(t),0,np.cos(t)]])
        elif roller==2:
            return np.array([[np.cos(t),np.sin(t),0],[-np.sin(t),np.cos(t),0],[0,0,1]])

    #Generate 3d grid or 4d tensor. Each grid represents a voxel. Each voxel represents the atom in it by onehot
    #Each complex in train set is rotated 9 times for data amplification.
    #The complexes in core set are not rotated.
    #The default resolution is 20*20*20.
    def grid(self,coords, features, resolution=1.0, max_dist=10.0, rotations=9):
        assert coords.shape[1] == 3
        assert coords.shape[0] == features.shape[0]


        grid=np.zeros((rotations+1,20,20,20,features.shape[1]),dtype=np.float32)
        x=y=z=np.array(range(-10,10),dtype=np.float32)+0.5
        for i in range(len(coords)):
            coord=coords[i]
            tmpx=abs(coord[0]-x)
            tmpy=abs(coord[1]-y)
            tmpz=abs(coord[2]-z)
            if np.max(tmpx)<=19.5 and np.max(tmpy)<=19.5 and np.max(tmpz) <=19.5:
                grid[0,np.argmin(tmpx),np.argmin(tmpy),np.argmin(tmpz)] += features[i]

        for j in range(rotations):
            theta = random.uniform(np.pi/18,np.pi/2)
            roller = random.randrange(3)
            coords = np.dot(coords, self.rotation_matrix(theta,roller))
            for i in range(len(coords)):
                coord=coords[i]
                tmpx=abs(coord[0]-x)
                tmpy=abs(coord[1]-y)
                tmpz=abs(coord[2]-z)
                if np.max(tmpx)<=19.5 and np.max(tmpy)<=19.5 and np.max(tmpz) <=19.5:
                    grid[j+1,np.argmin(tmpx),np.argmin(tmpy),np.argmin(tmpz)] += features[i]

        return grid

    def update_grid(self, grid, x, coords, features, resolution=1.0, max_dist=10.0, rotations=9):
        assert coords.shape[1] == 3
        assert coords.shape[0] == features.shape[0]
        y=z=x
        for i in range(len(coords)):
            coord=coords[i]
            tmpx=abs(coord[0]-x)
            tmpy=abs(coord[1]-y)
            tmpz=abs(coord[2]-z)
            if np.max(tmpx)<=19.5 and np.max(tmpy)<=19.5 and np.max(tmpz) <=19.5:
                grid[0,np.argmin(tmpx),np.argmin(tmpy),np.argmin(tmpz)] += features[i]

        for j in range(rotations):
            theta = random.uniform(np.pi/18,np.pi/2)
            roller = random.randrange(3)
            coords = np.dot(coords, self.rotation_matrix(theta,roller))
            for i in range(len(coords)):
                coord=coords[i]
                tmpx=abs(coord[0]-x)
                tmpy=abs(coord[1]-y)
                tmpz=abs(coord[2]-z)
                if np.max(tmpx)<=19.5 and np.max(tmpy)<=19.5 and np.max(tmpz) <=19.5:
                    grid[j+1,np.argmin(tmpx),np.argmin(tmpy),np.argmin(tmpz)] += features[i]

        return grid


def get_atom_features(atom, amino_acid, isprotein):
    ATOM_CODES = {}
    metals = ([3, 4, 11, 12, 13] + list(range(19, 32))
              + list(range(37, 51)) + list(range(55, 84))
              + list(range(87, 104)))
    atom_classes = [(5, 'B'), (6, 'C'), (7, 'N'), (8, 'O'), (15, 'P'), (16, 'S'), (34, 'Se'),
                ([9, 17, 35, 53], 'halogen'), (metals, 'metal')]
    for code, (atomidx, name) in enumerate(atom_classes):
        if type(atomidx) is list:
            for a in atomidx:
                ATOM_CODES[a] = code
        else:
            ATOM_CODES[atomidx] = code
    try:
        classes = ATOM_CODES[atom.GetAtomicNum()]
    except:
        classes = 9

    possible_chirality_list = [
        Chem.rdchem.ChiralType.CHI_UNSPECIFIED,
        Chem.rdchem.ChiralType.CHI_TETRAHEDRAL_CW,
        Chem.rdchem.ChiralType.CHI_TETRAHEDRAL_CCW,
        Chem.rdchem.ChiralType.CHI_OTHER
    ]
    chirality = possible_chirality_list.index(atom.GetChiralTag())

    possible_formal_charge_list = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
    try:
        charge = possible_formal_charge_list.index(atom.GetFormalCharge())
    except:
        charge = 11

    possible_hybridization_list = [
        Chem.rdchem.HybridizationType.S,
        Chem.rdchem.HybridizationType.SP,
        Chem.rdchem.HybridizationType.SP2,
        Chem.rdchem.HybridizationType.SP3,
        Chem.rdchem.HybridizationType.SP3D,
        Chem.rdchem.HybridizationType.SP3D2,
        Chem.rdchem.HybridizationType.UNSPECIFIED
    ]
    try:
        hyb = possible_hybridization_list.index(atom.GetHybridization())
    except:
        hyb = 6

    possible_numH_list = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    try:
        numH = possible_numH_list.index(atom.GetTotalNumHs())
    except:
        numH = 9

    possible_implicit_valence_list = [0, 1, 2, 3, 4, 5, 6, 7]
    try:
        valence = possible_implicit_valence_list.index(atom.GetTotalValence())
    except:
        valence = 8

    possible_degree_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    try:
        degree = possible_degree_list.index(atom.GetTotalDegree())
    except:
        degree = 11

    is_aromatic = [False, True]
    aromatic = is_aromatic.index(atom.GetIsAromatic())

    mass = atom.GetMass() / 100

    amino_acids = [
        'ALA', 'ARG', 'ASN', 'ASN', 'ASP', 'CYS', 'GLU', 'GLN', 'GLY', 'HIS', 'ILE', 'LEU', 'LYS', 'MET', 'PHE',
    ]
    if amino_acid in amino_acids:
      amino_acid = amino_acids.index(amino_acid)
    else:
      amino_acid = int(len(amino_acids) + 1)
```

```
        return [classes, chirality, charge, hyb, numH, valence, degree, aromatic, mass, amino_acid, isprotein]
        # return [classes, chirality, charge, hyb, numH, valence, degree, aromatic, amino_acid, isprotein]

    def get_min_max(compound_positions):
        minx,miny,minz = 999,999,999
        maxx,maxy,maxz = -999,-999,-999
        for pos in compound_positions:
            x, y, z = pos
            if x < minx:
                minx = x

            if y < miny:
                miny = y

            if z < minz:
                minz = z

            if x > maxx:
                maxx = x

            if y > maxy:
                maxy = y

            if z > maxz:
                maxz = z

        return (minx,miny,minz, maxx,maxy,maxz)

    def addNotUnique(nuC, uCL):
      x,y,z = nuC

      tmp1 = (x+1, y, z)
      if tmp1 not in uCL:
        uCL.append(tmp1)
        return tmp1

      tmp2 = (x, y+1, z)
      if tmp2 not in uCL:
        uCL.append(tmp2)
        return tmp2

      tmp3 = (x, y, z+1)
      if tmp3 not in uCL:
        uCL.append(tmp3)
        return tmp3

      tmp4 = (x+1, y+1, z)
      if tmp4 not in uCL:
        uCL.append(tmp4)
        return tmp4

      tmp5 = (x+1, y, z+1)
      if tmp5 not in uCL:
        uCL.append(tmp5)
        return tmp5

      tmp6 = (x, y+1, z+1)
      if tmp6 not in uCL:
        uCL.append(tmp6)
        return tmp6

      tmp7 = (x+1, y+1, z+1)
      if tmp7 not in uCL:
        uCL.append(tmp7)
        return tmp7

      tmp8 = (x-1, y, z)
      if tmp8 not in uCL:
        uCL.append(tmp8)
        return tmp8
```

```
     tmp9 = (x, y-1, z)
     if tmp9 not in uCL:
       uCL.append(tmp9)
       return tmp9

     tmp10 = (x, y, z-1)
     if tmp10 not in uCL:
       uCL.append(tmp10)
       return tmp10

     tmp11 = (x-1, y-1, z)
     if tmp11 not in uCL:
       uCL.append(tmp11)
       return tmp11

     tmp12 = (x-1, y, z-1)
     if tmp12 not in uCL:
       uCL.append(tmp12)
       return tmp12

     tmp13 = (x, y-1, z-1)
     if tmp13 not in uCL:
       uCL.append(tmp13)
       return tmp13

     tmp14 = (x-1, y-1, z-1)
     if tmp14 not in uCL:
       uCL.append(tmp14)
       return tmp14

   def setup_grid(protein_path, ligand_path, isprotein=1):
       compound = Chem.MolFromPDBFile(protein_path, False, False, 1)
       compound_conf = compound.GetConformer()
       compound_positions = compound_conf.GetPositions()

       result = get_min_max(compound_positions)

       atoms_aa = []
       with open(protein_path, 'r+') as f:
           readlines = f.readlines()
           f.close()
       for idx, lines in enumerate(readlines):
           if 'HETATM' in lines or 'ATOM' in lines:
               atoms_aa.append(lines[17:20])

       with open(ligand_path, 'r+') as f:
           readlines = f.readlines()
           f.close()
       for idx, lines in enumerate(readlines):
           if 'HETATM' in lines or 'ATOM' in lines:
               atoms_aa.append(lines[17:20])
       # print(result)
       # centerx = result[3] - result[0]
       # centery = result[4] - result[1]
       # centerz = result[5] - result[2]
       # print((centerx,centery,centerz))

       uniqueCoord = []
       # repeatedCoord = []

       atom_e = compound.GetAtomWithIdx(int(1))
       features_e = get_atom_features(atom_e, '', 1)
       grid=np.zeros((52,52,52,len(features_e)+3))
       # print(np.shape(grid))

       for idx, coords in enumerate(compound_positions):
         amino_acid = atoms_aa[idx]
         atom = compound.GetAtomWithIdx(int(idx))
         features = get_atom_features(atom, amino_acid, 1)
         # features.append(coords[0])
         # features.append(coords[1])
```

```
      # features.append(coords[2])
      features.extend([coords[0],coords[1],coords[2]])
      # print(idx)
      # print(coords)
      # print("-----------")
      x= coords[0] - (result[0] - 1)
      y= coords[1] - (result[1] - 1)
      z= coords[2] - (result[2] - 1)
      roundx = round(x)
      roundy = round(y)
      roundz = round(z)
      checkCoords = (roundx, roundy, roundz)
      # checkCoords2 = (x,y,z)
      if checkCoords not in uniqueCoord:
        uniqueCoord.append(checkCoords)
        grid[roundx, roundy, roundz] = features
      else:
        # repeatedCoord.append(checkCoords2)
        tmpCoord = addNotUnique(checkCoords, uniqueCoord)
        grid[tmpCoord[0], tmpCoord[1], tmpCoord[2]] = features

    con_com = len(compound_positions)
    compound = Chem.MolFromPDBFile(ligand_path, False, False, 1)
    compound_conf = compound.GetConformer()
    compound_positions = compound_conf.GetPositions()

    for idx, coords in enumerate(compound_positions):
      amino_acid = atoms_aa[idx+con_com]
      atom = compound.GetAtomWithIdx(int(idx))
      features = get_atom_features(atom, amino_acid, 0)
      features.extend([coords[0],coords[1],coords[2]])

      x= coords[0] - (result[0] - 1)
      y= coords[1] - (result[1] - 1)
      z= coords[2] - (result[2] - 1)
      roundx = round(x)
      roundy = round(y)
      roundz = round(z)
      checkCoords = (roundx, roundy, roundz)
      # checkCoords2 = (x,y,z)
      if checkCoords not in uniqueCoord:
        uniqueCoord.append(checkCoords)
        grid[roundx, roundy, roundz] = features
      else:
        # repeatedCoord.append(checkCoords2)
        tmpCoord = addNotUnique(checkCoords, uniqueCoord)
        grid[tmpCoord[0], tmpCoord[1], tmpCoord[2]] = features


    return grid
```

Get data batch

```
def get_data_batch(dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, index, type_model):
  core_grids=None
  core_ba= []
  core_stat= []
  ligand_list = os.listdir(ligand_folder)
  protein_list = os.listdir(protein_folder)
  label_list = os.listdir(label_folder)

  batch_list = [value for idx, value in enumerate(dataset_idx) if idx >= index * batch_size and idx < (index+1)*b
  if type_model == '3DCNN':
    for i in batch_list:
      # Feature = gridFromCenter()
      complexFile = ligand_list[i]

      # Get Data and Labels
      from_protein = complexFile.split('-')[0]
```

```
        complex_name = complexFile.split('.')[0]
        protein_path = os.path.join(protein_folder, from_protein+'.pdb')
        complexFile_path = os.path.join(ligand_folder, complexFile)

        # grid, minx, miny, minz = set_grid(protein_path)
        # grid = add_ligand(complexFile_path, grid, minx, miny, minz)
        grid = setup_grid(protein_path, complexFile_path)
        if core_grids is None:
            core_grids = []
        core_grids.append(grid)
        grid = []

        protein = [value for value in label_list if from_protein in value][0]
        label_file_path = os.path.join(label_folder, protein)

        df = pd.read_csv(label_file_path)
        listidx = df.index[df['file.pdb'] == complex_name].tolist()[0]
        ba = df['BA'][listidx]
        stat = df['Hit/No_hit'][listidx]
        if stat == 'hit':
          stat = 1
        else:
          stat = 0
        core_ba.append(ba)
        core_stat.append(stat)

      core_grids = np.array(core_grids)
      core_ba = np.array(core_ba)
      core_stat = np.array(core_stat)

      return core_grids, core_ba, core_stat


    if type_model == 'SFCNN':
      Feature = Feature_extractor()
      for id in batch_list:
        # if cur_batch*batch_size >=
        ligand_name = ligand_list[id]
        ligand_train_path = os.path.join(ligand_folder, ligand_name)
        # print(ligand_name)
        protein_name1 = ligand_name.split('-')[0]
        protein_name2 = protein_name1
        complex_name = ligand_name.split('.')[0]
        for name in protein_list:
          if protein_name1 in name:
            protein_name1 = name
            continue
        # print(protein_folder)
        protein_train_path = os.path.join(protein_folder, protein_name1)

        protein = Chem.MolFromPDBFile(protein_train_path, False, False, 0)
        ligand = Chem.MolFromPDBFile(ligand_train_path, False, False, 0)
        # train_complexes.append((protein, ligand))
        coords1, features1 = Feature.get_features(protein,1)
        coords2, features2 = Feature.get_features(ligand,0)
        protein = None
        ligand = None
        center=(np.max(coords2,axis=0)+np.min(coords2,axis=0))/2
        coords=np.concatenate([coords1,coords2],axis = 0)
        features=np.concatenate([features1,features2],axis = 0)
        assert len(coords) == len(features)
        coords = coords-center
        grid=Feature.grid(coords,features, rotations=0)
        if core_grids is None:
            core_grids = grid
        else:
            core_grids = np.concatenate([core_grids,grid],axis = 0)
        grid = []

        label_list = os.listdir(label_folder)
        for name in label_list:
```

```
        if protein_name2 in name:
          protein_name2 = name
          continue
      label_train_path = os.path.join(label_folder, protein_name2)
      df = pd.read_csv(label_train_path)
      listidx = df.index[df['file.pdb'] == complex_name].tolist()[0]
      ba = df['BA'][listidx]
      stat = df['Hit/No_hit'][listidx]
      if stat == 'hit':
        stat = 1
      else:
        stat = 0

      core_ba.append(ba)
      core_stat.append(stat)

    core_ba = np.array(core_ba)
    core_stat = np.array(core_stat)
    return core_grids, core_ba, core_stat
```

Other functions

```
def model_train(model, train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_p
  dataset_len = len(train_dataset_idx)
  runs = dataset_len // batch_size
  cur = 1
  log_txt = "log.txt"
  log_path = os.path.join(save_path, log_txt)
  readline = ''
  if os.path.exists(log_path):
    log_file = open(log_path,"r+")
    readline = log_file.readline()
    log_file.close()
  else:
    with open(log_path, 'w+') as f:
      f.write('0/'+str(runs))
      f.close()

  if readline == '' or int(readline.split('/')[0]) > runs or int(readline.split('/')[0]) == 0:
    cur = 1
    model.save(save_path)
  else:
    cur = int(readline.split('/')[0])
  check = 0

  print("----------------------- Start TrainDataset ------------------------")
  for i in range(int(cur-1),int(runs)):
    print("======================Batch "+ str(i+1)+"=============================")
    model = keras.models.load_model(save_path)
    print("Get dataset")
    gridList, baList, statList = get_data_batch(train_dataset_idx, protein_folder, ligand_folder, label_folder, b
    print("----------------------- Train TrainDataset ------------------------")
    model.fit(gridList, [baList, statList], epochs= epochs,verbose=0)
    gridList, baList, statList = [], [], []
    # PearsonR, MSE, RMSE, precision, recall, auc, f1_score, MCC = model_val_dataset(val_dataset_idx, protein_fol
    print("Save")
    model.save(save_path)
    log_file = open(log_path,"r+")
    readline = log_file.write(str(i)+'/'+str(runs))
    log_file.close()
    # if PearsonR > checkPearsonR and MCC > checkMCC and RMSE < checkRMSE:
    #   model.save(best_path)
    #   checkPearsonR = PearsonR
    #   checkMCC = MCC
    #   checkRMSE = RMSE
    if check == 0 and batch_size*i >= 2000:
      check +=1
```

```
        model.save(best_path)
      print("====================== End Batch "+ str(i+1)+"=============================")
    return model

  def SFCNN_model(input_shape=(20,20,20,28)):
    inp = Input(shape= input_shape, name='Input_Complexes')

    # Classification

    ## Check there are atoms
    x1 = Conv3D(7,kernel_size=(1,1,1),strides=(1,1,1))(inp)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(7,kernel_size=(3,3,3))(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    # x1 = AveragePooling3D(padding='same')(x1)

    x1 = Conv3D(56,kernel_size=(3,3,3),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    x1 = MaxPooling3D(pool_size=2)(x1)

    x1 = Conv3D(112,kernel_size=(3,3,3),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    x1 = MaxPooling3D(pool_size=2)(x1)

    x1 = Conv3D(224,kernel_size=(3,3,3),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    x1 = MaxPooling3D(pool_size=2)(x1)

    # Global Pooling
    x2 = GlobalAveragePooling3D()(x1)

    x2 = Dense(256)(x2)
    x2 = BatchNormalization()(x2)
    x2 = Activation('relu')(x2)
    x2 = Dropout(0.5)(x2)

    # Flattening
    x1 = Flatten()(x1)

    x1 = Dense(256)(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    x1 = Dropout(0.5)(x1)

    # Regression Output
    d1 = Dense(1,kernel_regularizer=tf.keras.regularizers.l2(0.01))(x1)
    # Classification Output
    d2 = Dense(1, activation='sigmoid')(x2)

    return Model(inputs=[inp], outputs=[d1,d2], name='Embedding')


  def CNN_model(drop_rate, input_shape= (52,52,52,14)):
    inp = Input(shape= input_shape, name='Input_Complexes')

    ## Check there are atoms
    ## Sketch the pattern of the whole biomolecule
    x1 = Conv3D(filters= 32, kernel_size=(1,1,1),strides=(1,1,1) ,padding='same', bias_initializer='zeros', kernel_
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 8, kernel_size=2, padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 8, kernel_size=3,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = MaxPooling3D(pool_size=2)(x1)

    ## Find pattern for chunk size

    x1 = Conv3D(filters= 128,kernel_size=(1,1,1),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 64,kernel_size=2,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 64,kernel_size=3,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = MaxPooling3D(pool_size=2)(x1)

    ## Find pattern for amino acid size
    x1 = Conv3D(filters= 256,kernel_size=(1,1,1),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 128,kernel_size=2,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 128,kernel_size=3,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = MaxPooling3D(pool_size=2)(x1)

    ## Find pattern for atom interaction size
    x1 = Conv3D(filters= 512,kernel_size=(1,1,1),padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 256,kernel_size=2,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    x1 = Conv3D(filters= 256,kernel_size=3,padding='same')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)

    # x1 = MaxPooling3D(pool_size=2)(x1)

    #   # Global Pooling
    x2 = GlobalAveragePooling3D()(x1)

    x2 = Dense(256)(x2)
    x2 = BatchNormalization()(x2)
    x2 = Activation('relu')(x2)
    x2 = Dropout(0.5)(x2)

    # # Flattening
    x1 = Flatten()(x1)

    x1 = Dense(256)(x1)
    x1 = BatchNormalization()(x1)
    x1 = Activation('relu')(x1)
    x1 = Dropout(0.5)(x1)

    # # Regression Output
    d1 = Dense(1,kernel_regularizer=tf.keras.regularizers.l2(0.01))(x1)
    # Classification Output
    d2 = Dense(1, activation='sigmoid')(x2)
```

```
  # return Model(inputs=[inp], outputs=[d1,d2], name='Embedding')
  return Model(inputs=[inp], outputs=[d1, d2], name='Embedding')


import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
import seaborn as sns



def plot_class(x, y):
  #create scatterplot with regression line
  # plt.scatter(x, y)
  # plt.show()
  # # sns.regplot(y_label, y_pred)
  new_y = []
  for value in y:
    if value >= 0.9:
      new_y.append(1)
    else:
      new_y.append(0)
  confusion_matrix = metrics.confusion_matrix(x, new_y)

  cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ["Hit", "No_H

  cm_display.plot()
  plt.show()
  return cm_display


def plot_reg(x, y):
  #create scatterplot with regression line
  #use green as color for individual points
  X = np.array(x)
  plt.plot(x, y, 'o', color='green')

  #obtain m (slope) and b(intercept) of linear regression line
  m, b = np.polyfit(x, y, 1)


  #use red as color for regression line
  plt.plot(X, m*X+b, color='red')



import math


def get_metrics(y_label, y_pred, ytype):

  if ytype == 0: # Regression
    print("+++++++++++++++++++++++++Regression+++++++++++++++++++++++++")
    PearsonR, p1 = pearsonr(y_label, y_pred)
    print('Pearson Correlation Coefficient: ' + str(PearsonR))
    print('P value: ' + str(p1))
    MSE = MeanAbsoluteError()
    MSE.update_state(y_label, y_pred)
    MSE = MSE.result().numpy()
    print('Mean Absolute Error: ' + str(MSE))

    rms = mean_squared_error(y_label, y_pred, squared=False)
    # rms = math.sqrt(mean_squared_error(y_label, y_pred, squared=False))
    print('Root Mean Error: ' + str(rms))

    r2 =r2_score(y_label, y_pred)
    print('Correlation of Covariance: ' + str(r2))

    rho, p2 = spearmanr(y_label, y_pred)
    print('Spearman Rank Correlation Coefficient: ' + str(rho))
    print('P value: ' + str(p2))
```

```
      # plot_reg(y_label, y_pred)

      return PearsonR, p1, MSE, rms, rho, p2

    if ytype == 1: # Classification
      print("++++++++++++++++++++++Classification++++++++++++++++++++++")

      tp = keras.metrics.TruePositives(thresholds= 0.9)
      tn = keras.metrics.TrueNegatives(thresholds= 0.9)
      fp = keras.metrics.FalsePositives(thresholds= 0.9)
      fn = keras.metrics.FalseNegatives(thresholds= 0.9)


      tp.update_state(y_label, y_pred)
      tn.update_state(y_label, y_pred)
      fp.update_state(y_label, y_pred)
      fn.update_state(y_label, y_pred)

      tp = tp.result().numpy()
      tn = tn.result().numpy()
      fp = fp.result().numpy()
      fn = fn.result().numpy()

      precision = tp/ (tp+fp) # PPV
      print('Precision: ' + str(precision))

      recall = tp/(tp+fn) # Recall - TPR
      print('Recall: ' + str(recall))

      specificity = tn/(tn+fp)
      print('Specificity: ' + str(specificity))

      NPV = tn/(tn+fn)
      print('NPV: ' + str(NPV))

      MCC = (tp*tn - fp*fn)/ math.sqrt(   (tp+fp)*(tp+fn)*(tn+fp)*(tn+fn)  ) # Phi coefficient
      print("Phi coefficient:" + str(MCC))

      return precision, recall, specificity, NPV, MCC

  def model_val_dataset(val_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path
    # dataset_len = len(val_dataset_idx)
    # runs = dataset_len // batch_size
    # model = keras.models.load_model(save_path)
    dataset_len = len(val_dataset_idx)
    runs = dataset_len // batch_size
    last_batch = dataset_len - batch_size*runs
    model = keras.models.load_model(save_path)
    # PearsonR_list, MCC_list, RSME_list = 0,0,0
    ba_Actual, stat_Actual, ba_Pred, stat_Pred = [], [], [], []

    print("----------------------- Start ValDataset ------------------------")
    for i in range(int(runs+1)):

      print("Get dataset on batch "+str(i+1))
      if i != runs+1:
        gridList, baList, statList = get_data_batch(val_dataset_idx, protein_folder, ligand_folder, label_folder, b
      else:
        gridList, baList, statList = get_data_batch(val_dataset_idx, protein_folder, ligand_folder, label_folder, l

      print("----------------------- Predict ValDataset ------------------------")
      result = model.predict(gridList, verbose=2)
      gridList = []
      pred_reg, pred_class = result

      baList = [value for value in baList.tolist()]
      statList = [value for value in statList.tolist()]

      pred_reg = [value[0] for value in pred_reg.tolist()]
      pred_class = [value[0] for value in pred_class.tolist()]
      if ba_Actual == []:
        ba_Actual = baList
        stat_Actual = statList
        ba_Pred = pred_reg
        stat_Pred = pred_class
      else:
        ba_Actual.extend(baList)
        stat_Actual.extend(statList)
        ba_Pred.extend(pred_reg)
        stat_Pred.extend(pred_class)

      baList, statList = [], []

    reg_res = get_metrics(ba_Actual, ba_Pred, 0)
    print("------------------------------------------------------------")
    class_res = get_metrics(stat_Actual, stat_Pred, 1)

    return ba_Actual, ba_Pred, stat_Actual, stat_Pred, reg_res, class_res

  def exportCSV(ligand_folder,pred_list,result,csv_path):
    fields = ['Model', 'Hit_Label', 'Hit_Prediction', 'BindingAffinity_Label', 'BindingAffinity_Prediction']
    rows = []
    ligand_list = os.listdir(ligand_folder)
    for idx, value in enumerate(pred_list):
      row = []
      row.append(ligand_list[value])
      row.append(result[0][idx])
      row.append(result[1][idx])
      row.append(result[2][idx])
      row.append(result[3][idx])

      rows.append(row)

    with open(csv_path, 'w') as csvfile:
      # creating a csv writer object
      csvwriter = csv.writer(csvfile)

      # writing the fields
      csvwriter.writerow(fields)

      # writing the data rows
      csvwriter.writerows(rows)
    csvfile.close()
```

Confirmed Scripts

```
  def hyper_train(train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path, be
    model = None
    if type_model == 'SFCNN':
      model = SFCNN_model()
    if type_model == '3DCNN':
      model = CNN_model(0.5)


    if model != None:
      batch_size, epochs, optimizer, loss = hyper_choices
      model.compile(optimizer= optimizer,
                    loss= loss)
      model_train(model, train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_p

  def exportVal(val_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path, best_p
    result = model_val_dataset(val_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, sa
    exportCSV(ligand_folder,val_dataset_idx,result,csv_path)
    return result

  # def test_model(train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path, b
  #   model = None
  #   if type_model == 'SFCNN':
```

```
#     model = SFCNN_model()
#   if type_model == '3DCNN':
#     model = CNN_model(0.5)


#   if model != None:
#     batch_size, epochs, optimizer, loss = hyper_choices
#     model.compile(optimizer= optimizer,
#                   loss= loss)
#     model.summary()
#     model.save('/content/temp')
#     for i in range(20):
#       print("Batch "+ str(i+1))
#       model = keras.models.load_model('/content/temp')
#       gridList, baList, statList = get_data_batch(train_dataset_idx, protein_folder, ligand_folder, label_folde
#       model.fit(gridList, [baList, statList], epochs= epochs,batch_size= batch_size,verbose=0)
#       gridList, baList, statList = [], [], []
#       model.save('/content/temp')
#     # model_train(model, train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, sa
#   return model

def test_model(train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path, bes
  model = None
  if type_model == 'SFCNN':
    model = SFCNN_model()
  if type_model == '3DCNN':
    model = CNN_model(0.5)


  if model != None:
    batch_size, epochs, optimizer, loss = hyper_choices
    model.compile(optimizer= optimizer,
                  loss= loss)
    model.summary()
#     model.save('/content/temp')
    for i in range(312):
      print("Batch "+ str(i+1))
#       model = keras.models.load_model('/content/temp')
      gridList, baList, statList = get_data_batch(train_dataset_idx, protein_folder, ligand_folder, label_folder,
      model.fit(gridList, [baList, statList], epochs= epochs,verbose=0)
      gridList, baList, statList = [], [], []
#       model.save('/content/temp')
#     # model_train(model, train_dataset_idx, protein_folder, ligand_folder, label_folder, batch_size, epochs, sa
  return model
```

Get dataset

```
p_folder = '/content/protein'
p_list = os.listdir(p_folder)

l_folder = '/content/ligand'
l_list = os.listdir(l_folder)

la_folder = '/content/label'
la_list = os.listdir(la_folder)

protein_test_path = '/content/protein/3qzq.pdb'

totalSize = len(l_list)
totalSize

permu = np.random.RandomState(seed=69).permutation(totalSize)

train_num, validate_num, test_num = 0,0,0
iDataset_num = totalSize
ratio = (60,20,20)

train_num = int(iDataset_num * (ratio[0]/ (ratio[0]+ratio[1]+ratio[2])))
val_num = int(iDataset_num * (ratio[1]/ (ratio[0]+ratio[1]+ratio[2])))
```

```
test_num = int(iDataset_num * (ratio[2]/ (ratio[0]+ratio[1]+ratio[2])))

val_num = 100
test_num = 500
last_num = 2000

train_list_IDs = permu[:train_num]
val_list_IDs = permu[train_num:(train_num+val_num)]
test_list_IDs = permu[(train_num+val_num):(train_num+val_num+test_num)]
last_list_IDs = permu[(train_num+val_num+test_num):(train_num+val_num+test_num+last_num)]
```

```
# permu = np.random.RandomState(seed=69).permutation(totalSize)
```

```
permu
```

```
array([ 2788, 12876,  5452, ..., 14740,  9818,  4041])
```

```
train_list_IDs
```

```
array([ 2788, 12876,  5452, ..., 13517, 15820, 11375])
```

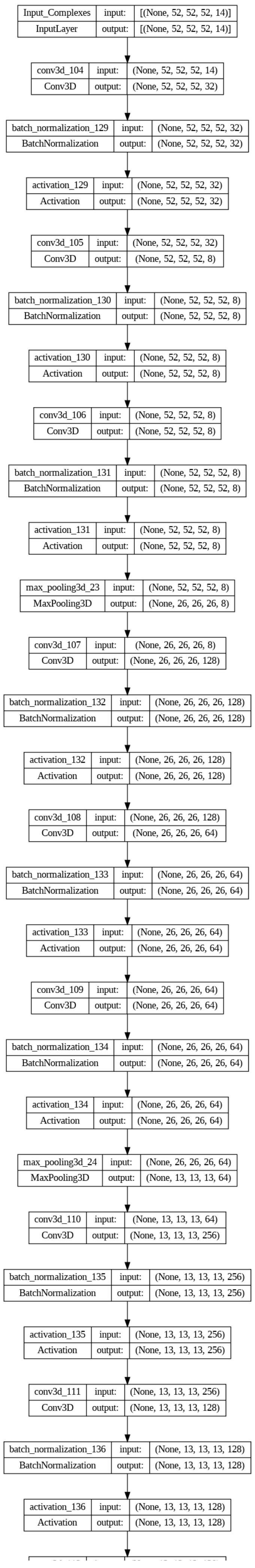
Setup Hyperparameters

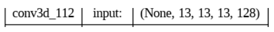
```
def cus_class_loss(y_label, y_pred):
    tp = keras.metrics.TruePositives(thresholds= 0.9)
    tn = keras.metrics.TrueNegatives(thresholds= 0.9)
    fp = keras.metrics.FalsePositives(thresholds= 0.9)
    fn = keras.metrics.FalseNegatives(thresholds= 0.9)


    tp.update_state(y_label, y_pred)
    tn.update_state(y_label, y_pred)
    fp.update_state(y_label, y_pred)
    fn.update_state(y_label, y_pred)

    tp = tp.result().numpy()
    tn = tn.result().numpy()
    fp = fp.result().numpy()
    fn = fn.result().numpy()
    loss_MCC = (tp*tn - fp*fn)/ math.sqrt(   (tp+fp)*(tp+fn)*(tn+fp)*(tn+fn)  )
    return loss_MCC

# train_dataset_idx,
# protein_folder, ligand_folder, label_folder, batch_size, epochs, save_path, best_path, type_model, hyper_choice



batch_size = 32
epochs = 200
optimizer=Adam(learning_rate=1e-4)
loss=['mean_squared_error', "binary_crossentropy"]

hypers = [batch_size, epochs, optimizer, loss]
# model.compile(optimizer= optimizer, loss= loss)

# batch_size, epochs, optimizer, loss = hyper_choices
model_type1 = "SFCNN"
save_path1 = '/content/gdrive/MyDrive/Final_Thesis/SFCNN/Save'
best_path1 = '/content/gdrive/MyDrive/Final_Thesis/SFCNN/Best'

model_type2 = "3DCNN"
save_path2 = '/content/gdrive/MyDrive/Final_Thesis/3DCNN/Save'
best_path2 = '/content/gdrive/MyDrive/Final_Thesis/3DCNN/Best'

# model_type2 = "3DCNN"
save_path3 = '/content/gdrive/MyDrive/Final_Thesis/test/Save'
```

```
best_path3 = '/content/gdrive/MyDrive/Final_Thesis/test/Best'

csv_path1_100 = '/content/gdrive/MyDrive/Final_Thesis/SFCNN/Report/resultSFCNN100.csv'
csv_path1_500 = '/content/gdrive/MyDrive/Final_Thesis/SFCNN/Report/resultSFCNN500.csv'
csv_path1_2000 = '/content/gdrive/MyDrive/Final_Thesis/SFCNN/Report/resultSFCNN2000.csv'

csv_path2_100 = '/content/gdrive/MyDrive/Final_Thesis/3DCNN/Report/result3DCNN100.csv'
csv_path2_500 = '/content/gdrive/MyDrive/Final_Thesis/3DCNN/Report/result3DCNN500.csv'
csv_path2_2000 = '/content/gdrive/MyDrive/Final_Thesis/3DCNN/Report/result3DCNN2000.csv'


csv_path3_100 = '/content/gdrive/MyDrive/Final_Thesis/test/Report/result3DCNN100.csv'
csv_path3_500 = '/content/gdrive/MyDrive/Final_Thesis/test/Report/result3DCNN500.csv'
csv_path3_2000 = '/content/gdrive/MyDrive/Final_Thesis/test/Report/result3DCNN2000.csv'
```

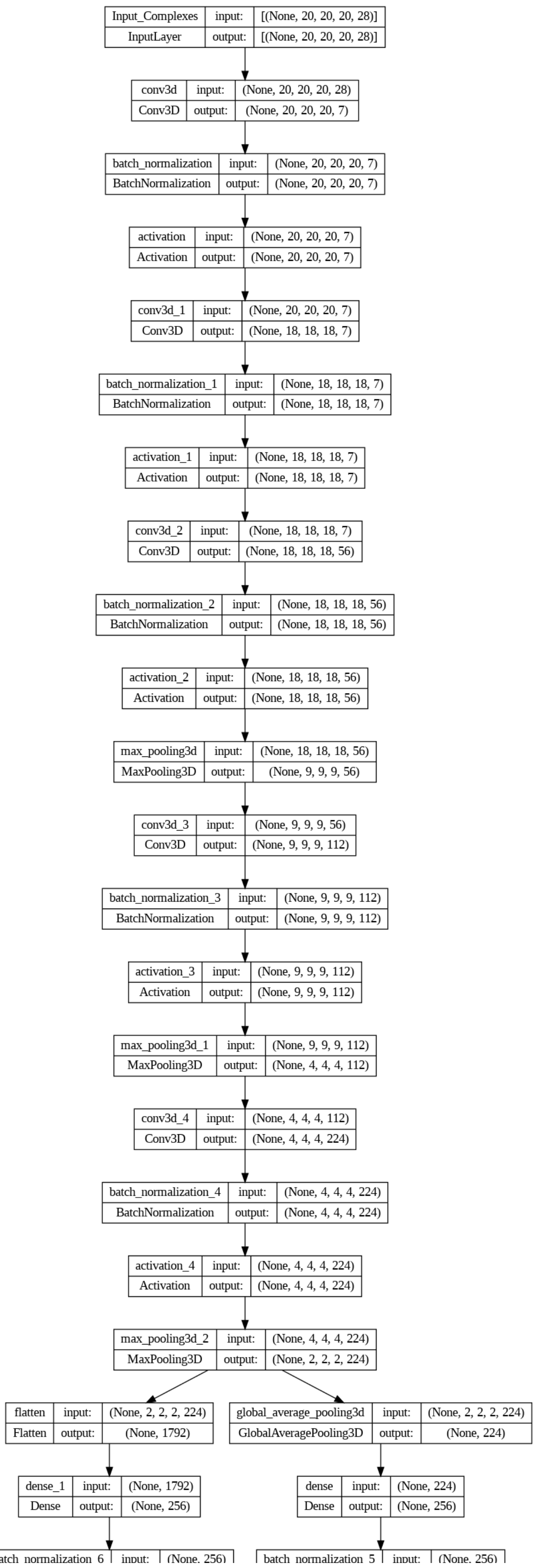
See Model Summary

```
from keras.utils.vis_utils import plot_model


test3DCNNModel = keras.models.load_model(save_path3)
# test3DCNNModel.summary()
plot_model(test3DCNNModel, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```


Input_Complexes input: [(None, 52, 52, 52, 14)]
InputLayer output: [(None, 52, 52, 52, 14)]
conv3d_104 input: (None, 52, 52, 52, 14)
Conv3D output: (None, 52, 52, 52, 32)
batch_normalization_129 input: (None, 52, 52, 52, 32)
BatchNormalization output: (None, 52, 52, 52, 32)
activation_129 input: (None, 52, 52, 52, 32)
Activation output: (None, 52, 52, 52, 32)
conv3d_105 input: (None, 52, 52, 52, 32)
Conv3D output: (None, 52, 52, 52, 8)
batch_normalization_130 input: (None, 52, 52, 52, 8)
BatchNormalization output: (None, 52, 52, 52, 8)
activation_130 input: (None, 52, 52, 52, 8)
Activation output: (None, 52, 52, 52, 8)
conv3d_106 input: (None, 52, 52, 52, 8)
Conv3D output: (None, 52, 52, 52, 8)
batch_normalization_131 input: (None, 52, 52, 52, 8)
BatchNormalization output: (None, 52, 52, 52, 8)
activation_131 input: (None, 52, 52, 52, 8)
Activation output: (None, 52, 52, 52, 8)
max_pooling3d_23 input: (None, 52, 52, 52, 8)
MaxPooling3D output: (None, 26, 26, 26, 8)
conv3d_107 input: (None, 26, 26, 26, 8)
Conv3D output: (None, 26, 26, 26, 128)
batch_normalization_132 input: (None, 26, 26, 26, 128)
BatchNormalization output: (None, 26, 26, 26, 128)
activation_132 input: (None, 26, 26, 26, 128)
Activation output: (None, 26, 26, 26, 128)
conv3d_108 input: (None, 26, 26, 26, 128)
Conv3D output: (None, 26, 26, 26, 64)
batch_normalization_133 input: (None, 26, 26, 26, 64)
BatchNormalization output: (None, 26, 26, 26, 64)
activation_133 input: (None, 26, 26, 26, 64)
Activation output: (None, 26, 26, 26, 64)
conv3d_109 input: (None, 26, 26, 26, 64)
Conv3D output: (None, 26, 26, 26, 64)
batch_normalization_134 input: (None, 26, 26, 26, 64)
BatchNormalization output: (None, 26, 26, 26, 64)
activation_134 input: (None, 26, 26, 26, 64)
Activation output: (None, 26, 26, 26, 64)
max_pooling3d_24 input: (None, 26, 26, 26, 64)
MaxPooling3D output: (None, 13, 13, 13, 64)
conv3d_110 input: (None, 13, 13, 13, 64)
Conv3D output: (None, 13, 13, 13, 256)
batch_normalization_135 input: (None, 13, 13, 13, 256)
BatchNormalization output: (None, 13, 13, 13, 256)
activation_135 input: (None, 13, 13, 13, 256)
Activation output: (None, 13, 13, 13, 256)
conv3d_111 input: (None, 13, 13, 13, 256)
Conv3D output: (None, 13, 13, 13, 128)
batch_normalization_136 input: (None, 13, 13, 13, 128)
BatchNormalization output: (None, 13, 13, 13, 128)
activation_136 input: (None, 13, 13, 13, 128)
Activation output: (None, 13, 13, 13, 128)



```
testSFCNNModel = keras.models.load_model(save_path1)
# testSFCNNModel.summary()
plot_model(testSFCNNModel, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```


Input_Complexes InputLayer input: [(None, 20, 20, 20, 28)] output: [(None, 20, 20, 20, 28)]
conv3d Conv3D input: (None, 20, 20, 20, 28) output: (None, 20, 20, 20, 7)
batch_normalization BatchNormalization input: (None, 20, 20, 20, 7) output: (None, 20, 20, 20, 7)
activation Activation input: (None, 20, 20, 20, 7) output: (None, 20, 20, 20, 7)
conv3d_1 Conv3D input: (None, 20, 20, 20, 7) output: (None, 18, 18, 18, 7)
batch_normalization_1 BatchNormalization input: (None, 18, 18, 18, 7) output: (None, 18, 18, 18, 7)
activation_1 Activation input: (None, 18, 18, 18, 7) output: (None, 18, 18, 18, 7)
conv3d_2 Conv3D input: (None, 18, 18, 18, 7) output: (None, 18, 18, 18, 56)
batch_normalization_2 BatchNormalization input: (None, 18, 18, 18, 56) output: (None, 18, 18, 18, 56)
activation_2 Activation input: (None, 18, 18, 18, 56) output: (None, 18, 18, 18, 56)
max_pooling3d MaxPooling3D input: (None, 18, 18, 18, 56) output: (None, 9, 9, 9, 56)
conv3d_3 Conv3D input: (None, 9, 9, 9, 56) output: (None, 9, 9, 9, 112)
batch_normalization_3 BatchNormalization input: (None, 9, 9, 9, 112) output: (None, 9, 9, 9, 112)
activation_3 Activation input: (None, 9, 9, 9, 112) output: (None, 9, 9, 9, 112)
max_pooling3d_1 MaxPooling3D input: (None, 9, 9, 9, 112) output: (None, 4, 4, 4, 112)
conv3d_4 Conv3D input: (None, 4, 4, 4, 112) output: (None, 4, 4, 4, 224)
batch_normalization_4 BatchNormalization input: (None, 4, 4, 4, 224) output: (None, 4, 4, 4, 224)
activation_4 Activation input: (None, 4, 4, 4, 224) output: (None, 4, 4, 4, 224)
max_pooling3d_2 MaxPooling3D input: (None, 4, 4, 4, 224) output: (None, 2, 2, 2, 224)
flatten Flatten input: (None, 2, 2, 2, 224) output: (None, 1792)
global_average_pooling3d GlobalAveragePooling3D input: (None, 2, 2, 2, 224) output: (None, 224)
dense_1 Dense input: (None, 1792) output: (None, 256)
dense Dense input: (None, 224) output: (None, 256)
batch_normalization_6 input: (None, 256)
batch_normalization_5 input: (None, 256)

```
def run_plots(result_model):
  plot_reg(result_model[0], result_model[1])
  plot_class(result_model[2], result_model[3])
```

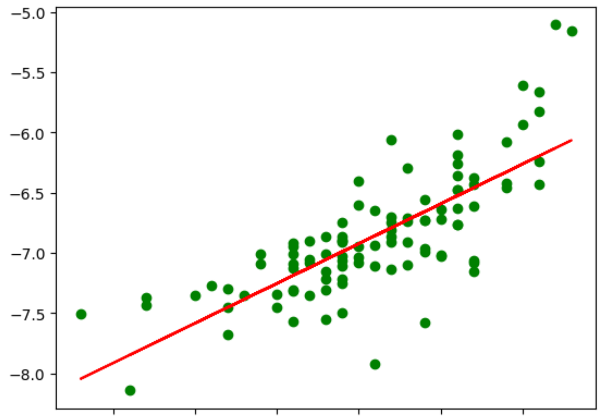
Run training SFCNN

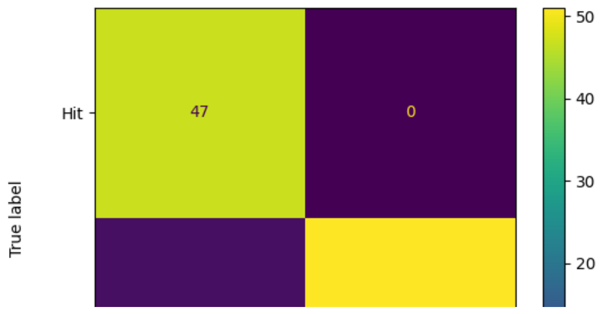
```
resultSFCNN1 = exportVal(val_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path1, best_path1,
```

```
    ----------------------- Start ValDataset -----------------------
    Get dataset on batch 1
    ----------------------- Predict ValDataset -----------------------
```

```
1/1 - 9s - 9s/epoch - 9s/step
Get dataset on batch 2
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 43ms/epoch - 43ms/step
Get dataset on batch 3
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 45ms/epoch - 45ms/step
Get dataset on batch 4
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 306ms/epoch - 306ms/step
+++++++++++++++++++++++++Regression+++++++++++++++++++++++++
Pearson Correlation Coefficient: 0.7610980650259652
P value: 3.952168548980092e-20
Mean Absolute Error: 0.30331326
Root Mean Error: 0.39754386183860047
Correlation of Covariance: 0.5622747975063129
Spearman Rank Correlation Coefficient: 0.758349610442936
P value: 6.434882061267643e-20
-------------------------------------------------------------
+++++++++++++++++++++++Classification+++++++++++++++++++++++
Precision: 1.0
Recall: 0.9622642
Specificity: 1.0
NPV: 0.9591837
Phi coefficient:0.9607226775452884
```

```
# plot_reg(resultSFCNN1[0], resultSFCNN1[1])
# plot_class(resultSFCNN1[2], resultSFCNN1[3])
run_plots(resultSFCNN1)
```



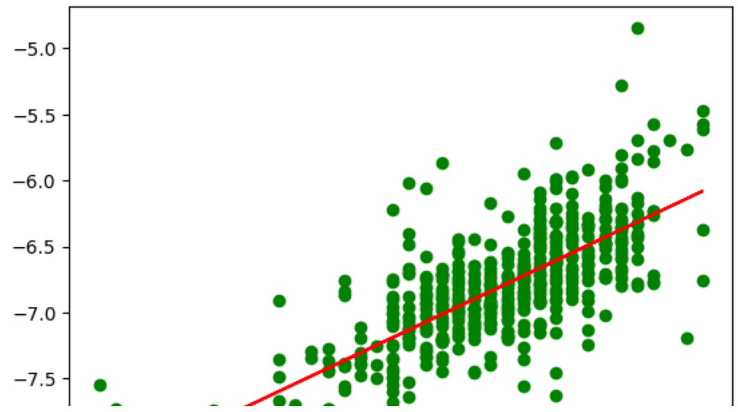


```
resultSFCNN2 = exportVal(test_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path1, best_path1
```

```
----------------------- Start ValDataset -----------------------
Get dataset on batch 1
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 199ms/epoch - 199ms/step
Get dataset on batch 2
```

```
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 44ms/epoch - 44ms/step
Get dataset on batch 3
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 34ms/epoch - 34ms/step
Get dataset on batch 4
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 5
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 6
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 7
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 33ms/epoch - 33ms/step
Get dataset on batch 8
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 9
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 10
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 11
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 41ms/epoch - 41ms/step
Get dataset on batch 12
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 13
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 14
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 30ms/epoch - 30ms/step
Get dataset on batch 15
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 54ms/epoch - 54ms/step
Get dataset on batch 16
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 345ms/epoch - 345ms/step
+++++++++++++++++++++++++Regression+++++++++++++++++++++++++
Pearson Correlation Coefficient: 0.7235470792816842
P value: 3.3565460058355834e-82
Mean Absolute Error: 0.30400905
Root Mean Error: 0.40510244561943526
Correlation of Covariance: 0.5125128343183774
Spearman Rank Correlation Coefficient: 0.7161012945011446
P value: 8.63352233792121e-80
```

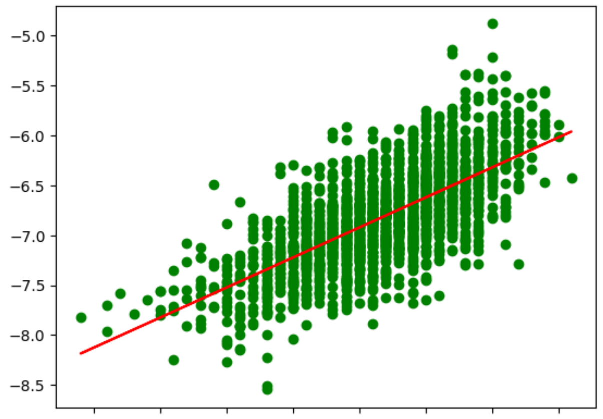
```
# plot_reg(resultSFCNN2[0], resultSFCNN2[1])
# plot_class(resultSFCNN2[2], resultSFCNN2[3])
run_plots(resultSFCNN2)
```

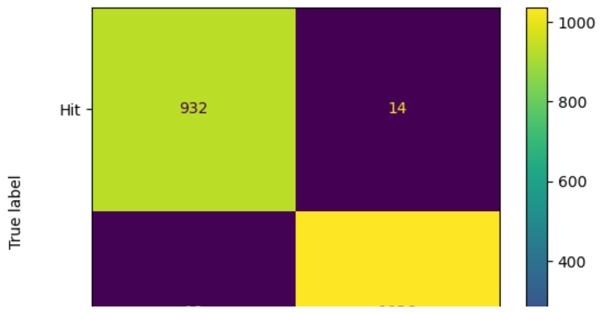


```
resultSFCNN3 = exportVal(last_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path1, best_path1
```

```
----------------------- Start ValDataset -----------------------
Get dataset on batch 1
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 188ms/epoch - 188ms/step
Get dataset on batch 2
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 33ms/epoch - 33ms/step
Get dataset on batch 3
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 33ms/epoch - 33ms/step
Get dataset on batch 4
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 34ms/epoch - 34ms/step
Get dataset on batch 5
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 41ms/epoch - 41ms/step
Get dataset on batch 6
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 7
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 8
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 9
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 10
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 11
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 12
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 13
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 14
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 31ms/epoch - 31ms/step
Get dataset on batch 15
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 16
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 17
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 33ms/epoch - 33ms/step
Get dataset on batch 18
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
Get dataset on batch 19
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 32ms/epoch - 32ms/step
```

```
# plot_reg(resultSFCNN3[0], resultSFCNN3[1])
# plot_class(resultSFCNN3[2], resultSFCNN3[3])
run_plots(resultSFCNN3)
```





```
# model_test = keras.models.load_model(save_path1)
# model_test.summary()
```

Run training 3DCNN

```
# hyper_train(train_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path3, best_path3, model_ty
```

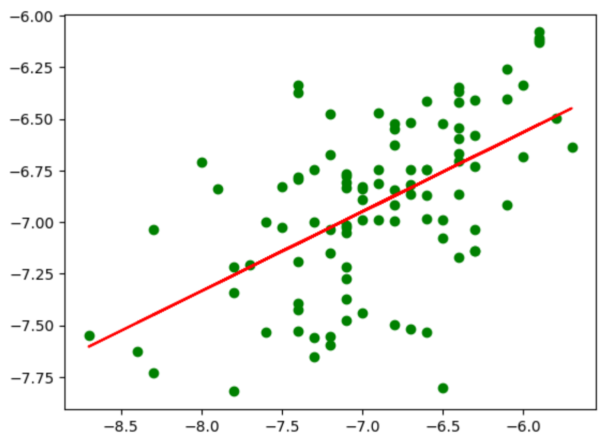
```
# exportVal(val_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path2, best_path2, model_type2,
```

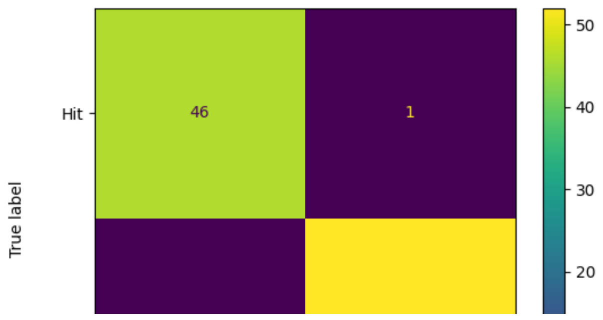
```
result3DCNN4 = exportVal(val_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path3, best_path3,
```

```
----------------------- Start ValDataset -----------------------
Get dataset on batch 1
----------------------- Predict ValDataset -----------------------
1/1 - 3s - 3s/epoch - 3s/step
Get dataset on batch 2
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 234ms/epoch - 234ms/step
Get dataset on batch 3
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 4
----------------------- Predict ValDataset -----------------------
1/1 - 1s - 814ms/epoch - 814ms/step
+++++++++++++++++++++++++Regression+++++++++++++++++++++++++
Pearson Correlation Coefficient: 0.5597116986644518
P value: 1.4141332800216367e-09
Mean Absolute Error: 0.3867036
Root Mean Error: 0.5039069835041357
Correlation of Covariance: 0.29671362764751497
Spearman Rank Correlation Coefficient: 0.5373963022576347
P value: 8.176348511231536e-09
-------------------------------------------------------------
+++++++++++++++++++++++Classification+++++++++++++++++++++++
Precision: 0.9811321
Recall: 0.9811321
Specificity: 0.9787234
```

```
NPV: 0.9787234
Phi coefficient:0.9598554797270172
```

```
run_plots(result3DCNN4)
```



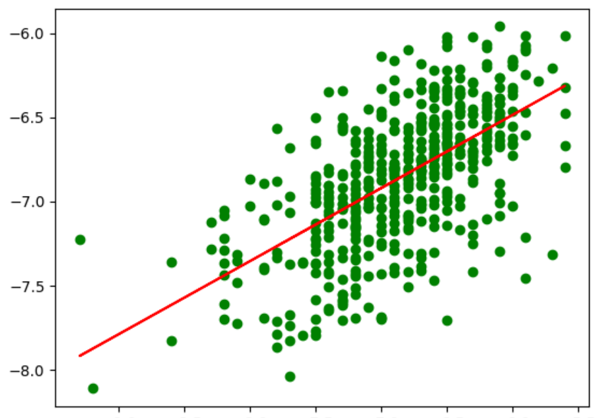


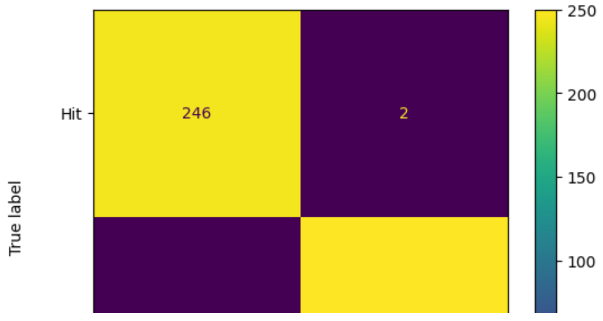
```
result3DCNN5 = exportVal(test_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path3, best_path3
```

```
----------------------- Start ValDataset -----------------------
Get dataset on batch 1
----------------------- Predict ValDataset -----------------------
WARNING:tensorflow:5 out of the last 68 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7a51d8fee29
1/1 - 0s - 498ms/epoch - 498ms/step
Get dataset on batch 2
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 238ms/epoch - 238ms/step
Get dataset on batch 3
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 231ms/epoch - 231ms/step
Get dataset on batch 4
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 228ms/epoch - 228ms/step
Get dataset on batch 5
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 228ms/epoch - 228ms/step
Get dataset on batch 6
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 7
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 231ms/epoch - 231ms/step
Get dataset on batch 8
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 9
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 233ms/epoch - 233ms/step
Get dataset on batch 10
----------------------- Predict ValDataset -----------------------
```

```
1/1 - 0s - 227ms/epoch - 227ms/step
Get dataset on batch 11
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 244ms/epoch - 244ms/step
Get dataset on batch 12
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 229ms/epoch - 229ms/step
Get dataset on batch 13
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 238ms/epoch - 238ms/step
Get dataset on batch 14
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 231ms/epoch - 231ms/step
Get dataset on batch 15
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 246ms/epoch - 246ms/step
Get dataset on batch 16
----------------------- Predict ValDataset -----------------------
1/1 - 2s - 2s/epoch - 2s/step
+++++++++++++++++++++++++Regression+++++++++++++++++++++++++
Pearson Correlation Coefficient: 0.6138496730781156
P value: 4.206029754108058e-53
Mean Absolute Error: 0.35416785
Root Mean Error: 0.4612898408042149
Correlation of Covariance: 0.3679066281801926
```

```
run_plots(result3DCNN5)
```



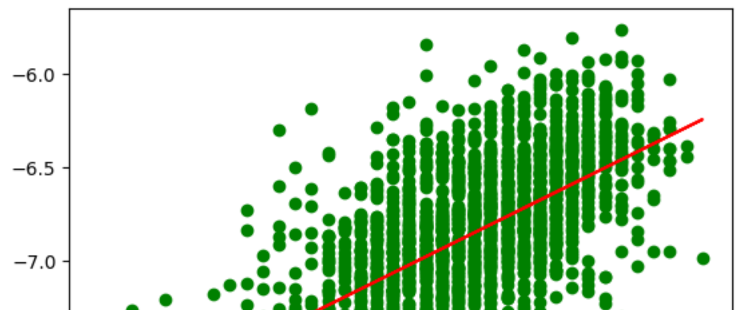


```
result3DCNN6 = exportVal(last_list_IDs, p_folder, l_folder, la_folder, batch_size, epochs, save_path3, best_path
```

```
----------------------- Start ValDataset -----------------------
Get dataset on batch 1
----------------------- Predict ValDataset -----------------------
1/1 - 1s - 511ms/epoch - 511ms/step
Get dataset on batch 2
----------------------- Predict ValDataset -----------------------
1/1 - 0s - 236ms/epoch - 236ms/step
```

```
Get dataset on batch 3
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 233ms/epoch - 233ms/step
Get dataset on batch 4
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 227ms/epoch - 227ms/step
Get dataset on batch 5
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 6
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 226ms/epoch - 226ms/step
Get dataset on batch 7
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 231ms/epoch - 231ms/step
Get dataset on batch 8
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 227ms/epoch - 227ms/step
Get dataset on batch 9
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 228ms/epoch - 228ms/step
Get dataset on batch 10
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 228ms/epoch - 228ms/step
Get dataset on batch 11
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 236ms/epoch - 236ms/step
Get dataset on batch 12
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 249ms/epoch - 249ms/step
Get dataset on batch 13
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 236ms/epoch - 236ms/step
Get dataset on batch 14
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 228ms/epoch - 228ms/step
Get dataset on batch 15
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 16
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 229ms/epoch - 229ms/step
Get dataset on batch 17
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 231ms/epoch - 231ms/step
Get dataset on batch 18
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 230ms/epoch - 230ms/step
Get dataset on batch 19
----------------------- Predict ValDataset ------------------------
1/1 - 0s - 228ms/epoch - 228ms/step
```

```
run_plots(result3DCNN6)
```



```
model_test = keras.models.load_model(save_path3)
model_test.summary()
```

```
Model: "Embedding"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 Input_Complexes (InputLayer)   [(None, 52, 52, 52,  0           []                               
                                 14)]                                                             
                                                                                                  
 conv3d_104 (Conv3D)            (None, 52, 52, 52,   480         ['Input_Complexes[0][0]']        
                                32)                                                               
                                                                                                  
 batch_normalization_129 (Batch  (None, 52, 52, 52,   128        ['conv3d_104[0][0]']             
 Normalization)                 32)                                                               
                                                                                                  
 activation_129 (Activation)    (None, 52, 52, 52,   0           ['batch_normalization_129[0][0]']
                                32)                                                               
                                                                                                  
 conv3d_105 (Conv3D)            (None, 52, 52, 52,   2056        ['activation_129[0][0]']         
                                8)                                                                
                                                                                                  
 batch_normalization_130 (Batch  (None, 52, 52, 52,   32         ['conv3d_105[0][0]']             
 Normalization)                 8)                                                                
                                                                                                  
 activation_130 (Activation)    (None, 52, 52, 52,   0           ['batch_normalization_130[0][0]']
                                8)                                                                
                                                                                                  
 conv3d_106 (Conv3D)            (None, 52, 52, 52,   1736        ['activation_130[0][0]']         
                                8)                                                                
                                                                                                  
 batch_normalization_131 (Batch  (None, 52, 52, 52,   32         ['conv3d_106[0][0]']             
 Normalization)                 8)                                                                
                                                                                                  
 activation_131 (Activation)    (None, 52, 52, 52,   0           ['batch_normalization_131[0][0]']
                                8)                                                                
                                                                                                  
 max_pooling3d_23 (MaxPooling3D  (None, 26, 26, 26,   0          ['activation_131[0][0]']         
 )                              8)                                                                
                                                                                                  
 conv3d_107 (Conv3D)            (None, 26, 26, 26,   1152        ['max_pooling3d_23[0][0]']       
                                128)                                                              
                                                                                                  
 batch_normalization_132 (Batch  (None, 26, 26, 26,   512        ['conv3d_107[0][0]']             
 Normalization)                 128)                                                              
                                                                                                  
 activation_132 (Activation)    (None, 26, 26, 26,   0           ['batch_normalization_132[0][0]']
                                128)                                                              
                                                                                                  
 conv3d_108 (Conv3D)            (None, 26, 26, 26,   65600       ['activation_132[0][0]']         
                                64)                                                               
                                                                                                  
 batch_normalization_133 (Batch  (None, 26, 26, 26,   256        ['conv3d_108[0][0]']             
 Normalization)                 64)                                                               
                                                                                                  
 activation_133 (Activation)    (None, 26, 26, 26,   0           ['batch_normalization_133[0][0]']
                                64)                                                               
                                                                                                  
 conv3d_109 (Conv3D)            (None, 26, 26, 26,   110656      ['activation_133[0][0]']         
                                64)                                                               
```