**1. 서버**

<MLP.py>

import pandas as pd  
import numpy as np  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras import optimizers  
  
np.random.seed(5)  
  
data = pd.read\_csv('RSSI-Dataset.csv')  
data['Cell'] = data['Cell'].replace(['A','B','C','D','E','F','G','H'],[0,1,2,3,4,5,6,7])  
X\_train = data[['AWifi', 'BWifi', 'CWifi']].values  
y\_train = data['Cell'].values  
from tensorflow.keras.utils import to\_categorical  
y\_train = to\_categorical(y\_train)  
model=Sequential()  
model.add(Dense(512, input\_dim=3, activation='relu'))  
model.add(Dense(512, activation='relu'))  
model.add(Dense(8, activation='softmax'))  
sgd=optimizers.SGD(lr=0.01)  
model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['acc'])  
hist=model.fit(X\_train, y\_train, batch\_size=10, epochs=400)  
class\_names = ['A','B','C','D','E','F','G','H']

<Server.py>

import socket  
from MLP import model, np, class\_names  
count = 0;  
host = '서버 운영중인 컴퓨터 ip'   
port = 9999 # Arbitrary non-privileged port  
server\_sock = socket.socket(socket.AF\_INET)  
server\_sock.bind((host, port))  
server\_sock.listen(1)  
print("기다리는 중")  
client\_sock, addr = server\_sock.accept()  
print('Connected by', addr)  
# 서버에서 "안드로이드에서 서버로 연결요청" 한번 받음  
data = client\_sock.recv(1024)  
print(data.decode("utf-8"), len(data))  
while (True):  
 data = client\_sock.recv(1024)  
 data1 = data.decode('utf-8')  
 data1 = data1.replace("[", "")  
 data1 = data1.replace("]", "")  
 data1 = data1.replace(",", "")  
 msp = data1.split()  
 input = list(map(int, msp))  
 count += 1  
 print('Received', input)  
 X\_test = np.array([input])  
 predictions = model.predict(X\_test)  
 result = class\_names[np.argmax(predictions)]  
 print(result)  
 client\_sock.send(result.encode())  
client\_sock.close()  
server\_sock.close()

**2. 어플리케이션**

<map.java>

public class map extends AppCompatActivity {  
 Timer timer;  
 TimerTask timerTask;  
 WifiManager wifiManager;  
 private List newRssi = new ArrayList(4);  
 boolean isPermitted = false;  
 final int MY\_PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION = 1;  
 private List<ScanResult> scanResultList;  
 int count = 0;  
 private String html = "";  
 private Handler mHandler;  
 private SocketChannel socket;  
 private Selector selector;  
 private sendDataThread SendData;  
 ImageView iv;  
 String data;  
 private String ip = "서버 ip"; // IP 번호   
 private int port = 9999; // port 번호  
 int pr,tp;  
 FusedLocationProviderClient fusedLocationProviderClient;  
 private static final int *REQUEST\_CODE* = 101;  
 BroadcastReceiver mReceiver = new BroadcastReceiver() {  
 @Override  
 public void onReceive(Context context, Intent intent) {  
 final String action = intent.getAction();  
 if (action.equals(WifiManager.*SCAN\_RESULTS\_AVAILABLE\_ACTION*)) {  
 getwifiInfo(); }  
 }  
 };  
  
 private void getwifiInfo() {  
 scanResultList = wifiManager.getScanResults();  
 for (int i = 0; i < scanResultList.size(); i++) {  
 ScanResult result = scanResultList.get(i);  
  
 if (result.SSID.equals("radioactivity") || result.SSID.equals("U+NetA990") || result.SSID.equals("olleh\_WiFi\_075B")) {  
 newRssi.add(result.level \* (-1));  
 if(result.SSID.equals("radioactivity")) {  
 pr = 1;  
 if(count == 2) {  
 Collections.*swap*(newRssi, 0, count);  
 Collections.*swap*(newRssi, count-1, count);  
 } else if(tp > pr) Collections.*swap*(newRssi, 0, count);  
 tp = pr;  
 } else if(result.SSID.equals("U+NetA990")) {  
 pr = 2;  
 if(tp > pr) Collections.*swap*(newRssi, count-1, count);  
 if(count == 2 && tp == 1) Collections.*swap*(newRssi, count-1, count);  
 tp = pr;  
 }  
 else {  
 pr = 3;  
 tp = pr;  
 }  
 count++;  
 if (count == 3) {  
 sendData(String.*valueOf*(newRssi));  
 count = 0;  
 newRssi.clear();  
 break;  
 }  
 }  
 }  
 }  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_map*);  
 fusedLocationProviderClient = LocationServices.*getFusedLocationProviderClient*(this);  
 requestRuntimePermission();  
 connect();  
 wifiManager = (WifiManager) getApplicationContext().getSystemService(*WIFI\_SERVICE*);  
 iv = (ImageView) findViewById(R.id.*cs*);  
  
 Handler hand = new Handler();  
 timerTask = new TimerTask() {  
 @Override  
 public void run() {  
 if (isPermitted) {  
 // wifi 스캔 시작  
 // wifi가 활성화되어있는지 확인 후 꺼져 있으면 켠다  
 if (wifiManager.getWifiState() != WifiManager.*WIFI\_STATE\_ENABLING*) {  
 wifiManager.setWifiEnabled(true);  
 }  
 wifiManager.startScan();  
 try {  
 ByteBuffer buffer = ByteBuffer.*allocate*(48);  
 int byteCount = socket.read(buffer);  
 buffer.flip();  
 Charset charset = Charset.*forName*("UTF-8");  
 data = charset.decode(buffer).toString();  
 Log.*w*("%d", data);  
 if (data.equals("C")) {  
 Intent goto\_menu = new Intent(map.this, work1.class);  
 startActivity(goto\_menu);  
 } else {

iv.setImageResource(R.drawable.*cursur*);  
 if (data.equals("A")) {  
 iv.setX(32);  
 iv.setY(150);  
 iv.invalidate();  
 } else if (data.equals("B")) {  
 iv.setX(385);  
 iv.setY(150);  
 iv.invalidate();  
 } else if (data.equals("D")) {  
 iv.setX(760);  
 iv.setY(150);  
 iv.invalidate();  
 } else if (data.equals("E")) {  
 iv.setX(32);  
 iv.setY(510);  
 iv.invalidate();  
 } else if (data.equals("F")) {  
 iv.setX(385);  
 iv.setY(510);  
 iv.invalidate();  
 } else if (data.equals("G")) {  
 iv.setX(580);  
 iv.setY(510);  
 iv.invalidate();  
 } else if (data.equals("H")) {  
 iv.setX(760);  
 iv.setY(510);  
 iv.invalidate();  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 Toast.*makeText*(getApplicationContext(),  
 "Location access 권한이 없습니다..", Toast.*LENGTH\_LONG*).show();  
 }  
 }  
 };  
 timer = new Timer();  
 timer.schedule(timerTask, 0, 1000);  
 }  
  
  
 protected void connect() {  
 mHandler = new Handler();  
 Log.*w*("connect", "연결 하는중");  
 // 받아오는거  
 Thread checkUpdate = new Thread() {  
 public void run() {  
 // 서버 접속  
 try {  
 setSocket(ip, port);  
 Log.*w*("서버 접속됨", "서버 접속됨");  
 } catch (IOException e1) {  
 Log.*w*("서버접속못함", "서버접속못함");  
 e1.printStackTrace();  
 }  
 Log.*w*("edit 넘어가야 할 값 : ", "안드로이드에서 서버로 연결요청");  
 sendData("안드로이드에서 서버로 연결요청");  
 Log.*w*("버퍼", "버퍼생성 잘됨");  
 }  
 };  
 // 소켓 접속 시도, 버퍼생성  
 checkUpdate.start();  
 }  
 private void setSocket(String ip, int port) throws IOException {  
 selector = Selector.*open*();  
 socket = SocketChannel.*open*(new InetSocketAddress(ip, port));  
 socket.configureBlocking(false);  
 socket.register(selector, SelectionKey.*OP\_CONNECT* | SelectionKey.*OP\_READ* | SelectionKey.*OP\_WRITE*);  
 }  
  
 public void sendData(String data) {  
 SendData = new sendDataThread(socket, data);  
 SendData.start();  
 }  
  
 public class sendDataThread extends Thread {  
 private SocketChannel sdt\_hSocketChannel;  
 private String data;  
  
 public sendDataThread(SocketChannel sc, String d) {  
 sdt\_hSocketChannel = sc;  
 data = d;  
 }  
  
 public void run() {  
 try { // 데이터 전송.  
 sdt\_hSocketChannel.write(ByteBuffer.*wrap*(data.getBytes()));  
 } catch (Exception e1) {  
  
 }  
 }  
 }  
  
 @Override  
 protected void onResume() {  
 super.onResume();  
 // wifi scan 결과가 나왔을 때 전송되는 broadcast를 받기 위해  
 // IntentFilter 객체를 생성하고 이를 이용하여 BroadcastReceiver 객체를 등록한다  
 IntentFilter filter = new IntentFilter(WifiManager.*SCAN\_RESULTS\_AVAILABLE\_ACTION*);  
 registerReceiver(mReceiver, filter);  
  
 }  
  
 @Override  
 protected void onPause() {  
 super.onPause();  
 unregisterReceiver(mReceiver);  
 }  
  
  
 private void requestRuntimePermission() {  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 // Runtime permission check  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 if (ContextCompat.*checkSelfPermission*(map.this,  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*)  
 != PackageManager.*PERMISSION\_GRANTED*) {  
  
 // Should we show an explanation?  
 if (ActivityCompat.*shouldShowRequestPermissionRationale*(map.this,  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*)) {  
  
 // Show an expanation to the user \*asynchronously\* -- don't block  
 // this thread waiting for the user's response! After the user  
 // sees the explanation, try again to request the permission.  
  
 } else {  
  
 // No explanation needed, we can request the permission.  
  
 ActivityCompat.*requestPermissions*(map.this,  
 new String[]{Manifest.permission.*ACCESS\_FINE\_LOCATION*},  
 MY\_PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION);  
 }  
 } else {  
 // ACCESS\_FINE\_LOCATION 권한이 있는 것  
 isPermitted = true;  
 }  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 }  
  
 @Override  
 public void onRequestPermissionsResult(int requestCode,  
 String permissions[], int[] grantResults) {  
 switch (requestCode) {  
 case MY\_PERMISSIONS\_REQUEST\_ACCESS\_FINE\_LOCATION: {  
 // If request is cancelled, the result arrays are empty.  
 if (grantResults.length > 0  
 && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
  
  
  
 // ACCESS\_FINE\_LOCATION 권한을 얻음  
 isPermitted = true;  
  
 } else {  
  
  
  
 // 권한을 얻지 못 하였으므로 location 요청 작업을 수행할 수 없다  
 // 적절히 대처한다  
 isPermitted = false;  
  
 }  
 return;  
 }  
  
 // other 'case' lines to check for other  
 // permissions this app might request  
 }  
 }  
}