**南昌大学信息工程学院，2023级，新一代电子信息技术电子2班 信号检测理论期末考试题目**

姓名：赖宝林 班级： 新一代电子信息技术

要求：从以下五题中任意选择一题，结合信号检测理论涉及到的基本概念，完成文献综述、方案设计、仿真验证和总结分析，并形成不少于30页的研究报告。研究报告及验证代码的最终提交时间不晚于2024年1月15日。

**题目1**：**基于功率谱熵的信号存在性检测以及信号起止位置检测：**

要求：

1、 调制方式至少包含：AM、FM、FSK、BPSK、16QAM、MSK、8CPSK； 2、 调制信号过 AWGN 信道后的信噪比覆盖：-30dB-0dB；

3、 生成至少100段信号，至少10段没有信号，总采样点数1024点，信号的占空比10-90%； 4、 模拟调制的符号速率：AM、FM，不高于4kSPS；

5、 数字调制的符号速率：10MSPS；

6、 模拟调制信号的采样率：1MSPS；

7、 数字调制信号的采样率：40MSPS。

完成信号存在性检测以及信号起止时间的检测，统计检测器的 ROC 性能及 MSE，并完成的 课题研究和仿真报告。

**题目2**：**通信信号的调制方式识别方法：**

要求：

1、 调制方式至少包含：AM、FM、FSK、BPSK、16QAM、MSK、8CPSK； 2、 调制信号过 AWGN 信道后的信噪比覆盖：-30dB-0dB；

3、 生成至少100段信号，至少10段没有信号，总采样点数1024点，信号的占空比10-90%； 4、 模拟调制的符号速率：AM、FM，不高于4kSPS；

5、 数字调制的符号速率：10MSPS；

6、 模拟调制信号的采样率：1MSPS；

7、 数字调制信号的采样率：40MSPS。

针对常见通信信号的调制方式识别问题，总结现存的统计量（特征量），进行仿真验证，统 计检测器的 ROC 性能及 MSE，完成研究报告。

**题目3**：**目标跟踪：**

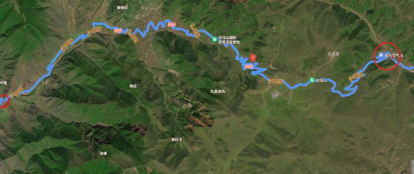
要求：

1、 以 G318左贡段为例，包含急上坡、急下坡、急拐弯、平直路线； 2、 假设 GNSS 信号时有时无，精度在10米，10秒到5分钟内，随机出现 GNSS 信号，每次 出现维持1分钟；

3、 根据车载的速度表、加速度表、陀螺仪，已知车辆的速度和方向。 4、 没有离线地图，除已知起点 A 和终点 B 的 GNSS 坐标外，另外，假设已知不多于20个关 键途经点的 GNSS 位置，途经点位置自定义。

建立 GNSS、速度表、加速度表等误差模型，使用卡尔曼滤波器的改进型算法，融合 GNSS、 速度表、加速度表和陀螺仪，请实现从起点到终点的车辆导航，并将跟踪出的轨迹显示到地 图上，显示航迹质量曲线及分析稳定时间及质量的影响因素。

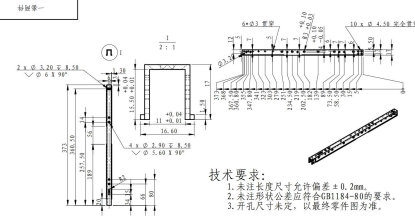
建议使用 Python。



**题目4**：**螺孔及转角检测**：

要求：利用图像处理，实现下图结构件中螺孔数量、位置、尺寸的检测；实现转角数量、位 置、尺寸的检测。

统计检测器的 ROC 性能及 MSE，完成研究报告。



**题目5**：**通信信号载波检测及跟踪：**

1、 调制方式：QPSK；

2、 调制信号过 AWGN 信道后的信噪比：0dB；

3、 符号速率：10MSPS；

4、 载波频偏及多普勒：20kHz；

5、 载波多普勒变化率：20kHz/s；

6、 采样率：40MSPS；

7、 时隙长度：2ms；

8、 占空比：46%；

9、 载波中心频率：100MHz。

调研检测和跟踪通信载波频率偏差、相位差的常见统计量及方法，分析影响稳定时间的因素， 得出载波频率偏差和相位差的跟踪曲线，完成仿真和研究报告。

**目录**

[文献综述 5](#_Toc154740565)

[**1.1研究意义 5**](#_Toc154740566)

[**1.2 国内外研究现状 5**](#_Toc154740567)

[二、方案设计 7](#_Toc154740568)

[**2.1题目分析与要求提炼 7**](#_Toc154740569)

[**2.2 仿真方案设计 7**](#_Toc154740570)

[**2.3 数据集选择 7**](#_Toc154740571)

[三、仿真实验验证 7](#_Toc154740572)

[**3.1实验目的 7**](#_Toc154740573)

[**3.2 实验预期 7**](#_Toc154740574)

[**3.3实验步骤 7**](#_Toc154740575)

[**3.3.1环境搭建 7**](#_Toc154740576)

[**3.3.2 程序设计 7**](#_Toc154740577)

[**3.3.3 模型训练 7**](#_Toc154740578)

[**3.3.4 模型评估 7**](#_Toc154740579)

[四、总结分析 7](#_Toc154740580)

**基于多模型融合的调制方式识别仿真研究**

# 文献综述

## 1.1研究意义

随着无线通信技术的发展，人们的交流与合作越发密切，对于通信质量和性能的要求不断提升。在非合作通信中，接收方无法知晓接收信号的调制方式， 若想要进行正确解调，需要先提取目标信号并识别其调制方式。随着通信体制逐渐复杂化，调制方式逐渐多样化，发送方与接收方之间的通信协议开销增大，也对接收方的自主检测和调制方式识别能力提出了更高的要求。在民用领域中，信号存在性检测和调制识别可以应用到干扰检测、频谱管理等场景中，更高效地利用稀缺的频谱资源；在军事领域中，现代化战争正朝着高度信息化的方向发展，截获、破解敌方的通信信号，获得敌方信息并干扰敌方通信，已经成为通信对抗和信息战的制胜关键之一。因此，信号存在性检测与调制识别在军民应用中都有着重要的实际意义和广泛的应用场景。

## 1.2 国内外研究现状

早期的调制识别依赖于有经验的工程师进行人工识别，消耗大量的人力物力，也带来了准确率不稳定、效率较低的问题。为解决上述问题，人们开始尝试研究自动化、智能化的调制识别技术，并取得了一定的成效。Weaver 等人于 1969 年第一次提出了自动调制识别的概念，从此自动调制识别的研究开始走进国内外学者的视野，自动调制识别领域的研究成果也不断涌现。

近年来，自动调制识别的研究主要分为以下三类：基于似然函数的调制方式识别方法、基于特征的调制方式识别方法和基于深度学习的调制方式识别方法。

（1）基于似然函数的调制方式识别方法

Kim 等人在 1988 年提出了平均似然比检验方法（Average Likelihood Ratio Test，ALRT），将信道未知参数看作随机变量，对二进制相移键控（Binary Phase Shift Keying，BPSK）与正交相移键控（Quadrature Phase Shift Keying，QPSK）信号进行识别，这也标志着以贝叶斯决策理论为核心的基于似然函数的调制识别方法正式形成。基于似然函数的方法是一个多重假设检验问题，基本思想是先对待识别信号的检验统计量进行分析推导，建立适用于当前识别任务的损失函数，设置相应的门限，再基于贝叶斯准则进行判决。

（2）基于特征的调制方式识别方法

与基于似然函数的方法相比，基于特征的调制识别方法可以达到次优的效果，在大大减少计算复杂度的同时，通过选取表征能力强的特征组合与拟合能力强的分类器，来逼近识别性能上界，逐渐成为调制识别领域研究的主流。基于特征的调制识别方法主要分为两步：第一，对接收到的信号进行预处理与特征提取；第二，将提取到的特征送入分类器进行识别和分类。特征提取过程中，所选取的特征要具有代表性和准确性，既能表征出各类调制信号的独特性，又能体现不同调制方式之间的差异性。常用的调制识别特征有信号的瞬时特征（瞬时振幅、瞬时相位和瞬时频率等）、高阶统计量（高阶矩和累积量等）特征、小波变换特征、循环谱特征和星座图特征等等。

（3）基于深度学习的调制方式识别方法

近年来，深度学习在各个领域发展迅速，在自动调制识别领域也取得了一定成效。基于深度学习的方法不需要信号的先验知识，也不需要进行人工特征的提取，在网络结构合适且训练数据量充足的情况下，可以达到比传统分类器更好的效果。

CNN（卷积神经网络）可以对图像模型进行特征提取并进行分类，（长短时神经网络）LSTM作为RNN（循环神经网络）的改进，能学习到时间序列中长时间的依赖，并减少梯度爆炸问题。

基于深度学习的调制识别方法利用深度学习模型的学习能力提取信号中的自适应特征，在数据量充足的情况下，可以获得比其他传统方法更好的识别效果。但也存在着一些问题：第一，深度学习模型提取的自适应特征存在着冗余性和不确定性，在信噪比较低或调制种类增多时，特征的表征能力有待提升，与人工特征相结合可能获得更好的效果；第二，深度学习模型需要充足的有标签数据集，这在实际场景中较为缺乏，因此现有研究普遍缺乏实验验证，无法证明其在实际通信环境中的可行性。

# 二、方案设计

## 2.1题目分析与要求提炼

根据题目要求，我们要在接收端实现对初采样信号的调制方式分类，此时的信号不可避免地携带了加性噪声，信号衰弱，多普勒频偏等信道效应，甚至需要考虑发射端本振的LO漂移。在这些干扰下的接收信号往往由人工经验难以辨别，此时利用近些年发展迅速的机器学习算法对其进行分类将是一个恰到好处的办法。针对题目要求的各族调制信号，我们采用的数据集要包含常见的各种调制方式，涵盖调频，调相，调幅，以及新型QAM调制，各族内也要包含若干种易混淆的调制方式，同时涵盖不同信噪比，以适应真实场景下的通信情况，同时以此来提高分类器的分类性能和鲁棒性。而机器学习分类器的原理都是基于数据特征来进行区分的，分类器的输入特征越显著，边界越清晰，分类性能就越好。基于此，可以概况出题目的核心要求是分类信号满足调制方式的多样性以及不同信噪比，核心办法是提取接收原始采样信号的调制方式特征，设置合适的机器学习分类器，并在此基础上尝试进一步优化达到当前研究现状下性能指标。而题目中关于采样速率和采样长度，则尽可能模拟实际通信中的参数，以增加实际场景下的应用意义。

## 2.2 仿真方案设计

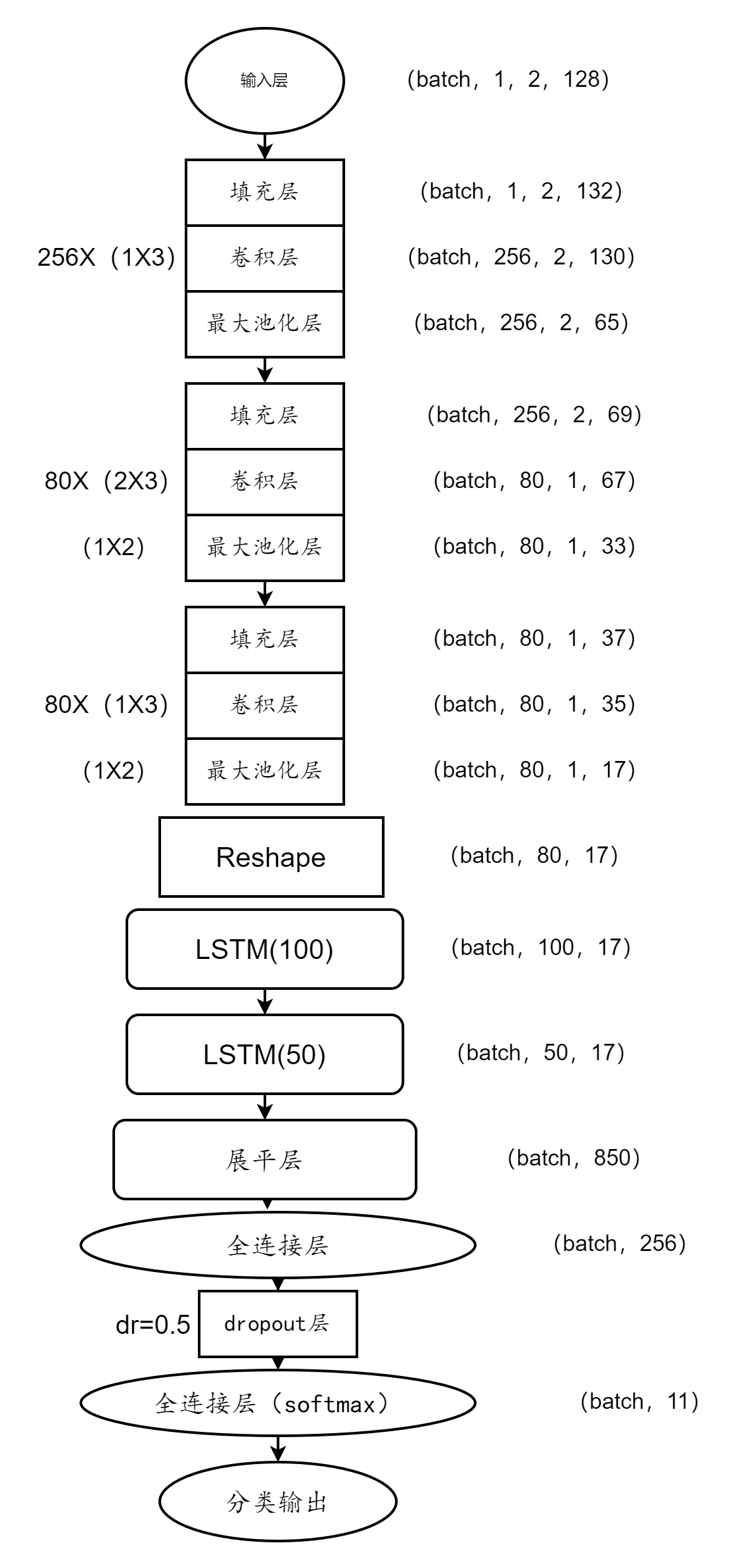


图2.1A路卷积神经网络提取自适应特征

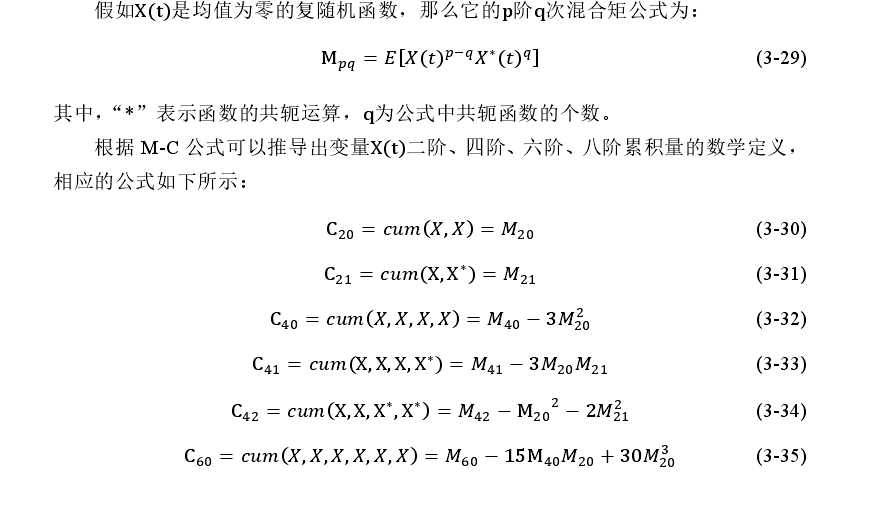


图2.2B路高阶混合矩阵计算公式

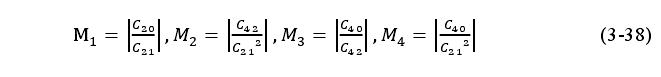


图2.3 B路由混合矩构建参数

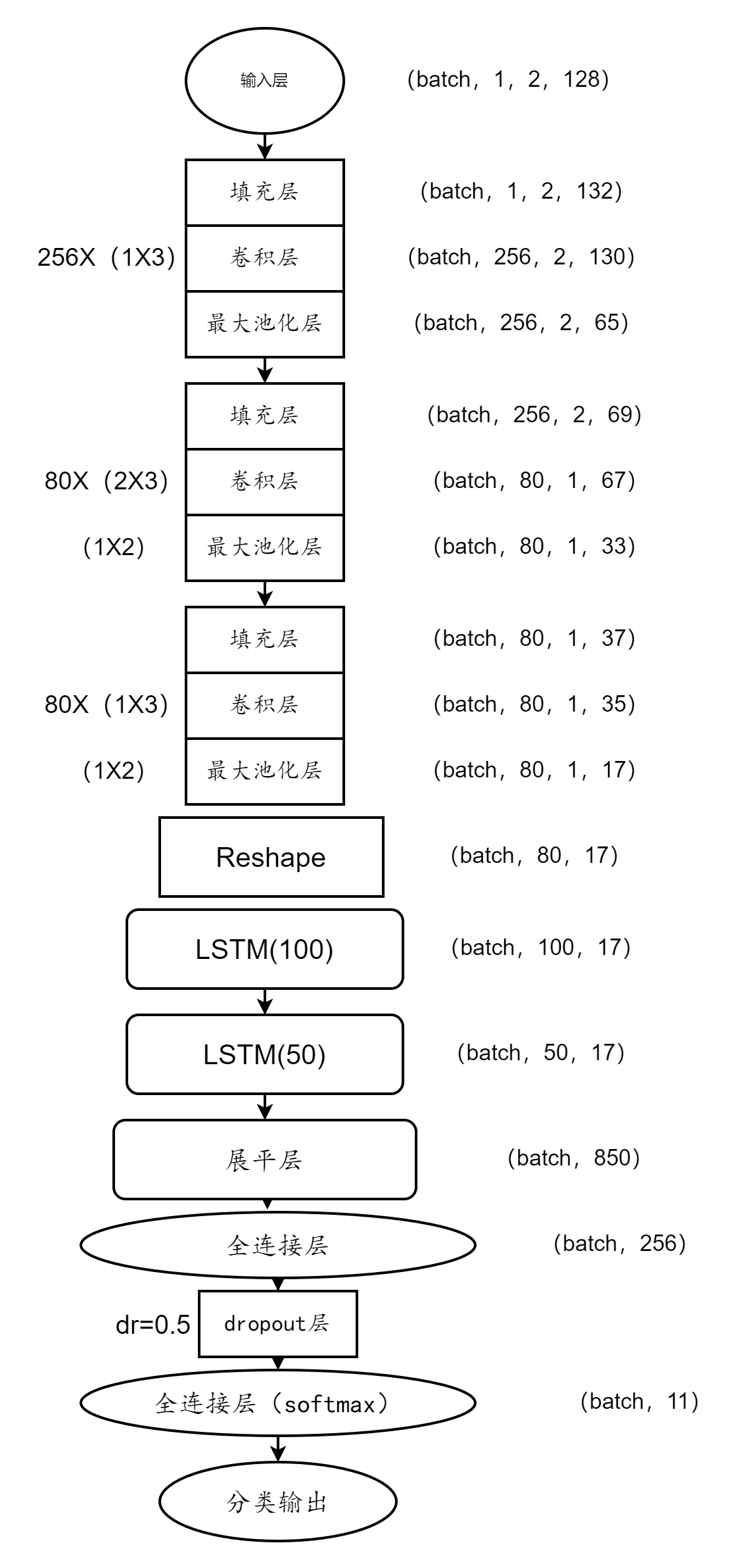


图2.4AB路特征参数组合再统一输入全连接层

## 2.3 数据集选择

鉴于通信调制信号的调制方式特征主要体现在其星座图上，所以我们在接收端对接收信号进行采样，采样后进行分量提取分解成同相和正交两股分量，便可得到星座图数据。

为了满足实验要求，选取近些年该领域通用的数据集，来自OMnisig工作室提供的Radio ML2016.10a 实验数据集。该数据集共包含 220000 条信号样本，每条信号样本为有 128 个采样点的I/Q数据，有 11种调制信号 8PSK、AM-DSB、AM-SSB、BPSK、CPFSK、GFSK、PAM4、QAM16、QAM64、QPSK 和 WBFM，每种调制信号在[-20，18]d B 范围内每隔 2d B 生成 1000条样本。该数据集是利用开源软件无线电平台 GNU Radio 生成，在产生过程中采用了大量真实语音信号，并采用了 GNU Radio 中动态信道模型模拟了许多信道效应，包括频偏、相偏、高斯白噪声、频率选择性衰落等。

# 三、仿真实验验证

### 3.1实验目的

提取给定数据集下信号的调制方式特征，配置合适的机器学习分类器，实现不同信噪比下接收信号的自动调制识别。

### 3.2 实验预期

当前最新的国内外研究表明，基于卷积神经网络的自动调制识别准确度最高，在同Radio ML2016.10a数据集下已经可以达到95以上的准确度。但是论文上只能参考到原理和方法，对算法实现的代码并不开源，所以在编程实现和优化这一块需要原创性发挥，这也是本实验的重点。

### 3.3实验步骤

1.搭建ubuntu18.04 + python3.8 + tensorflow-gpu + cuda的支持GPU加速的深度学习开发平台。

2.搭建卷积神经网络，应用卷积神经网络在图像特征提取中的突出表现，提取数据集的自适应特征（神经网络的反向传播采用的随机梯度下降算法是一种自适应算法，故常称卷积神经网络所提取的数据特征为自适应特征）。

3.编写高阶累积量算法，提取数据集的人工特征（运用统计学或者各种先验知识设计算法，计算出的特征向量称为人工特征）。

4.用复合网络结合2，3步的两种特征，合并输入全连接层进行分类。

#### **3.3.1环境搭建**

Linux环境配置较为复杂，为方便大家复现，在此介绍windows下最简易的环境搭建：

1. python3.8安装

python新版本与tensorflow库容易冲突，故选择稳定性高的老版本。

1. python创建虚拟环境

python -n vevn MyNet

1. 激活已创建的虚拟环境

MyNet/Scripts/activate.bat

1. 安装必备的库（一键操作）

pip install -r requirements.txt

1. 运行训练程序

python filename.py

gpu加速环境搭建：

1. 安装对应型号GPU的驱动
2. 下载并安装cuda
3. 在cuda中添加cudnn
4. 添加相应环境变量C:\ProgramFiles\NVIDIAGPUComputingToolkit\CUDA\v10.1\extras\CUPTI\lib64 &PATH

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\cudnn\bin &PATH

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\include &PATH

1. 运行训练程序即可启用gpu加速

#### **3.3.2 程序设计**

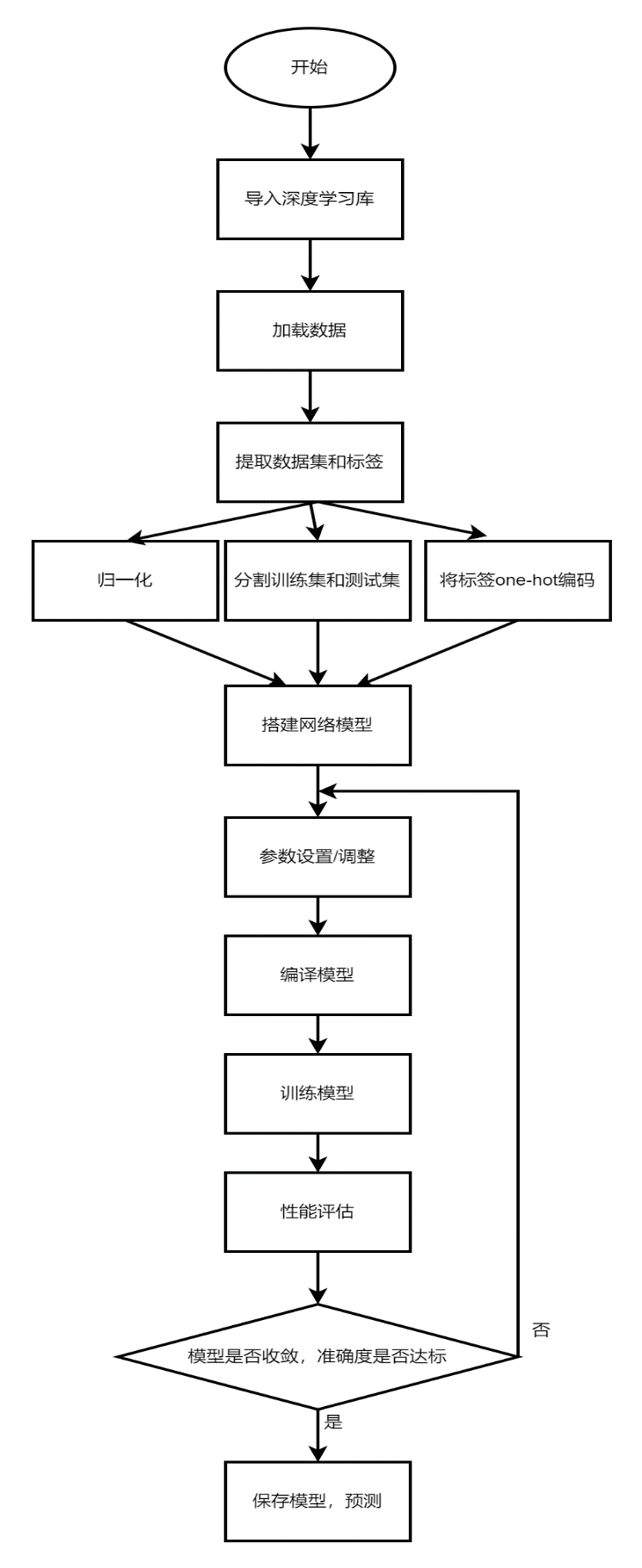


图3.1程序设计思路

#### **3.3.3 模型训练**

模型的训练过程主要有网络权重和偏置参数以正态分布初始化，数据前向传播，计算损失，通过梯度下降法进行反向调参，再使用测试集验证评估模型性能，每个过程都按预设的批处理量规定的数据量进行，用上所有训练数据则算作一次训练迭代。

网络权重的偏置参数是在训练过程中自动调整的，然而另有一些需要提前设定的参数，这些参数称为超参数，通常有学习率，迭代次数，批量大小，神经网络结构，以及激活函数和优化器的选择。这些都要根据经验选择最适合于相关任务的配置。本次分类任务，超参数配置如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学习率 | 迭代次数 | 批量大小 | 激活函数 | 损失函数 | 优化器 |
| 0.01 | 100，早期停止 | 512 | RELU | 交叉熵 | Adam |

表3.1网络超参数设置

每迭代一定轮数，参数的更新速度将会减慢，此时将学习率η减半，使模型能提取到样本中的更多信息，提升模型的训练速度。同时，若损失函数loss在一定轮数内基本不变，则说明模型已经收敛，终止训练并保存此时的最佳网络参数（早期停止策略）。

#### **3.3.4 模型评估**

对应神经网络性能的评估方式主要有混淆矩阵，预测准确率曲线，和训练损失曲线，其中前两者主要评估网络的预测性能，而损失曲线主要评估模型收敛速度，损失最低值也能说明网络学习数据特征的一个能力。一下为本次网络的损失下降曲线。

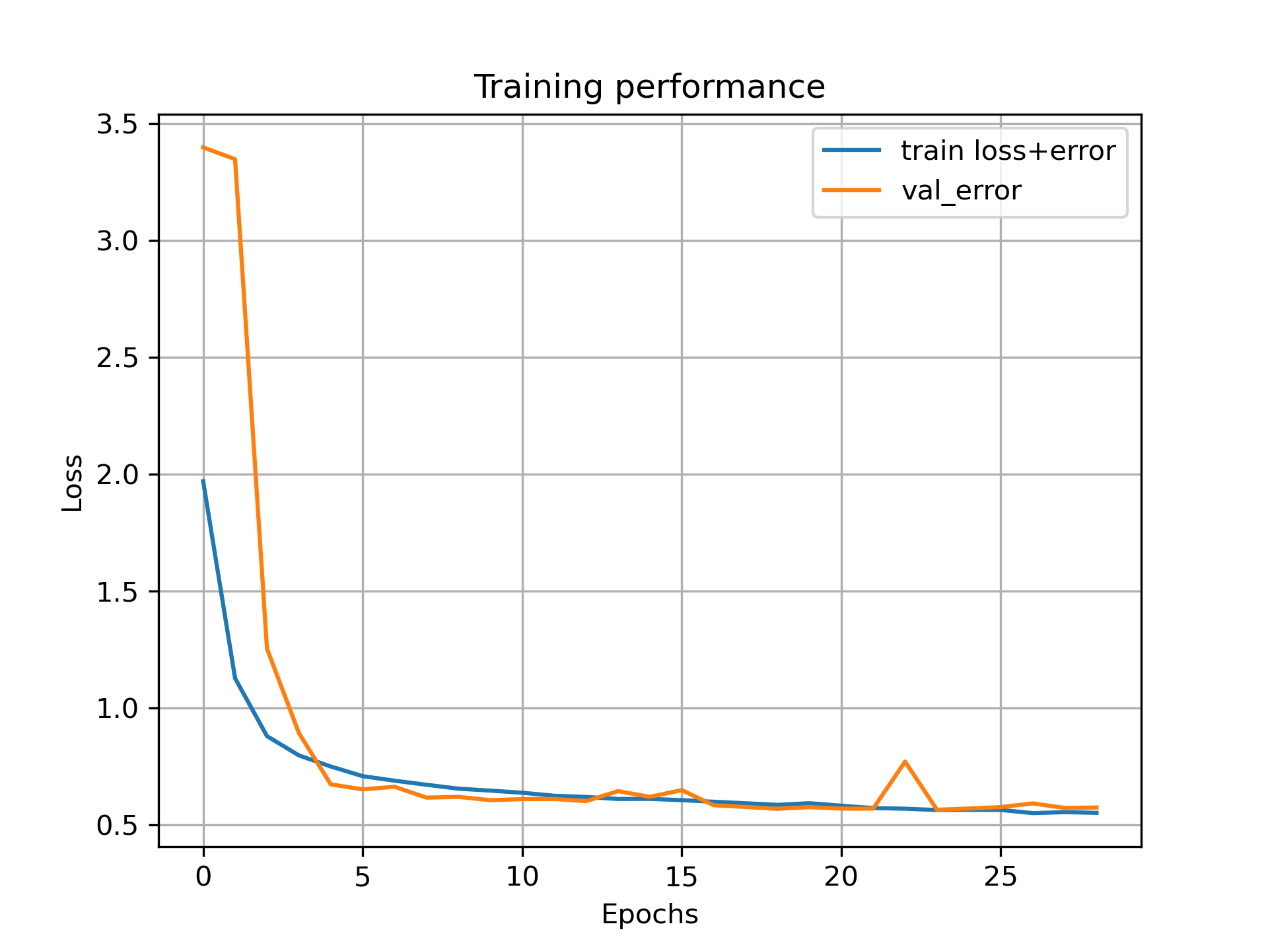


图3.2损失函数收敛曲线

从曲线可以看到，模型损失函数在5轮迭代内就接近收敛，模型的收敛速度较快，反映出网络的分类性能优良，对数据的特征敏感，同时，可以观察到在5轮迭代之后曲线逐渐平滑，说明随着迭代次数的增加，参数调整减缓，但仍然朝减小损失的方向继续调整，在达到饱和值之前，注意到测试集的预测损失曲线在20次后有小幅度的回升尖峰，这其实是正常现象，由于测试集按批次进行验证，批次选择512，学习率选择0.01，在小批次下由于数据的特异性，出现小幅波动再回降是正常现象。

以下为整体混淆矩阵和各信噪比下混淆矩阵

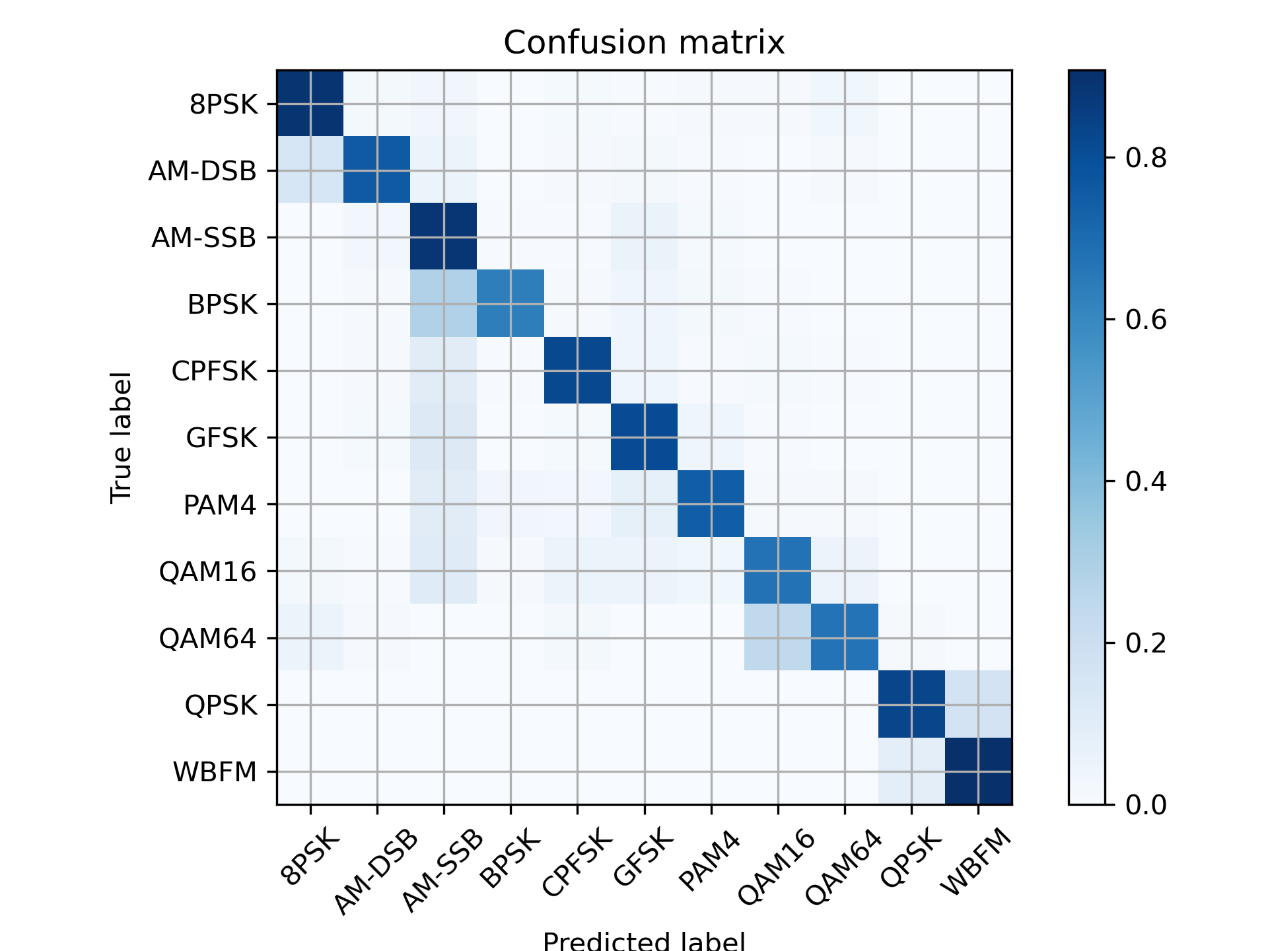
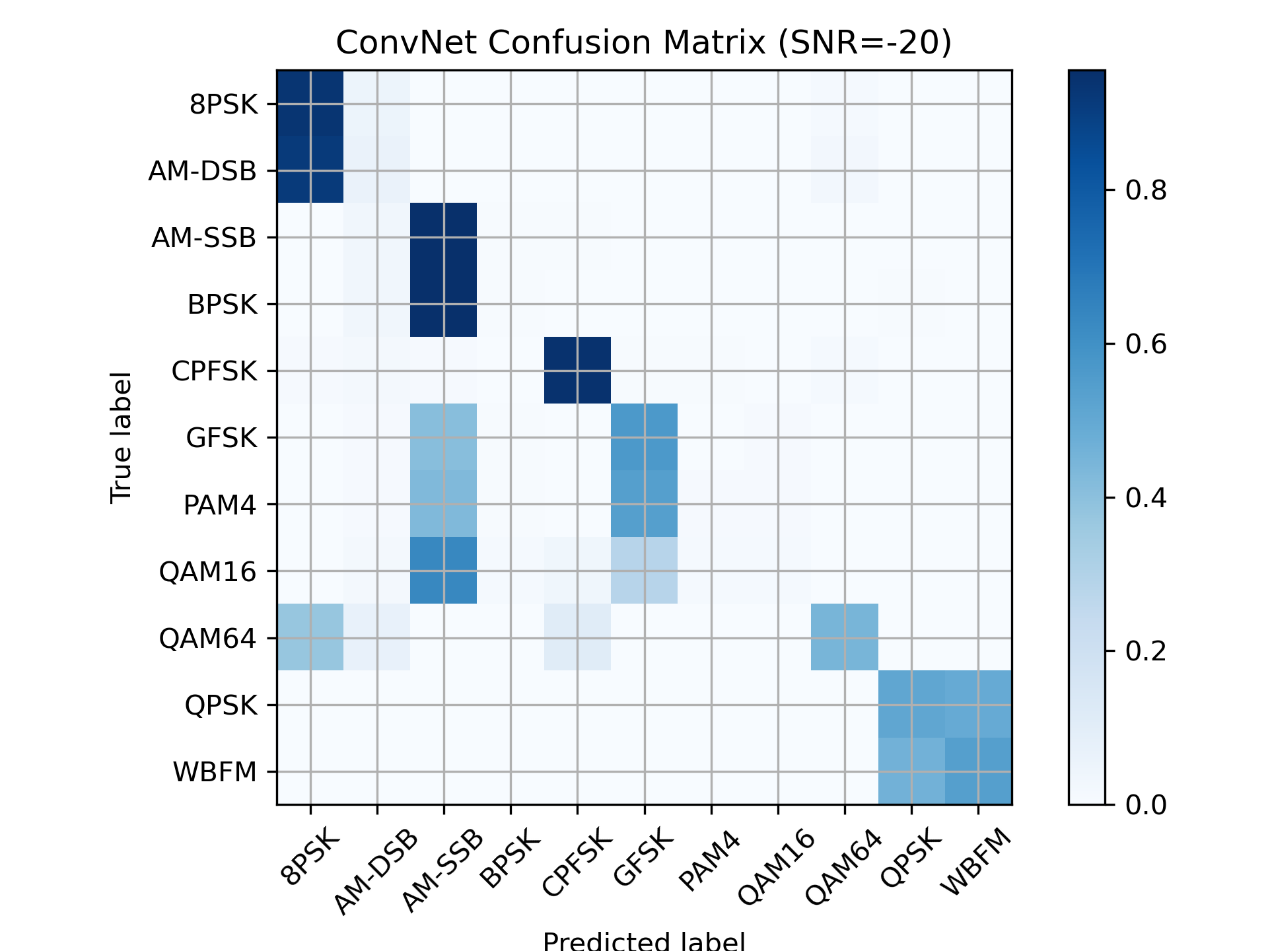
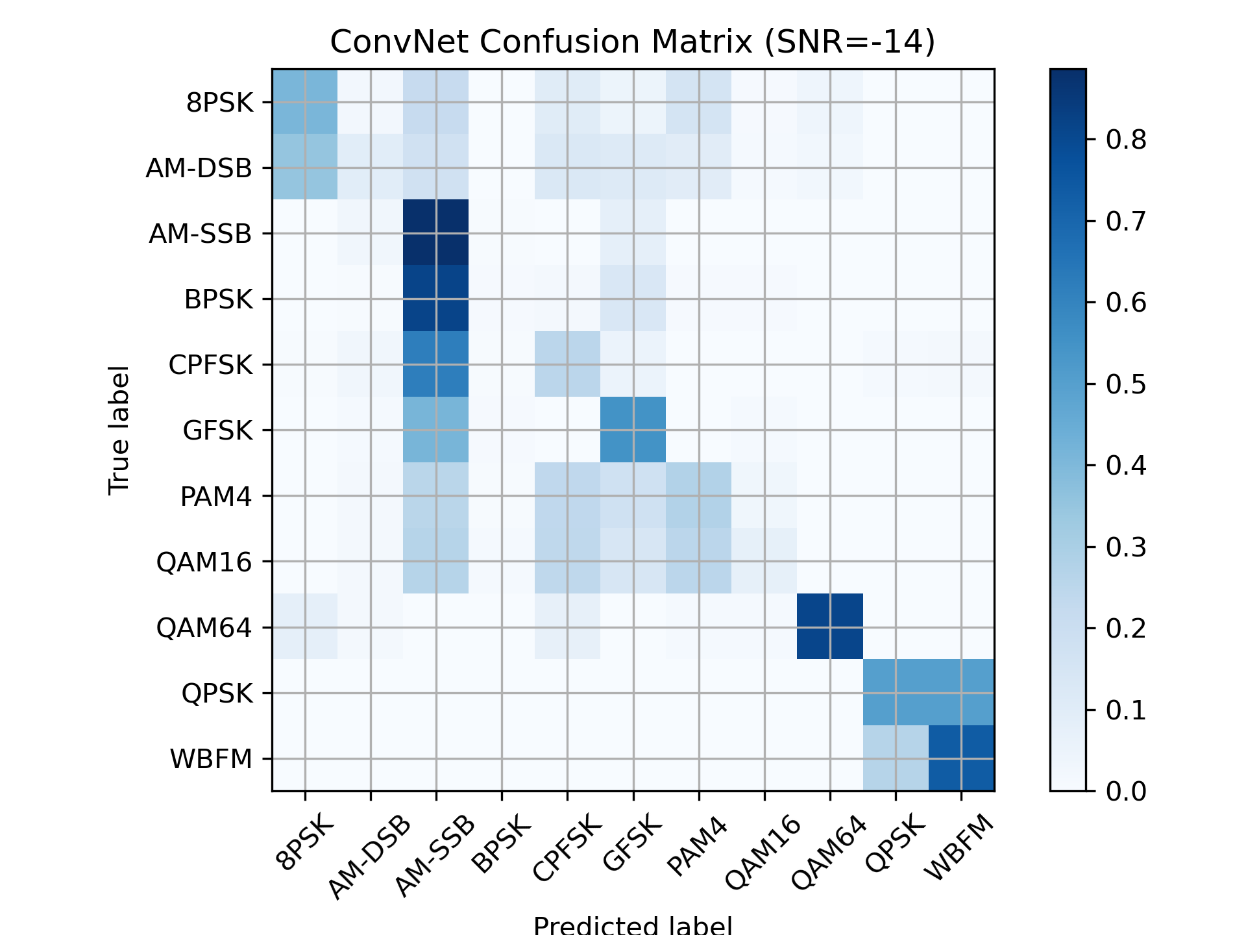
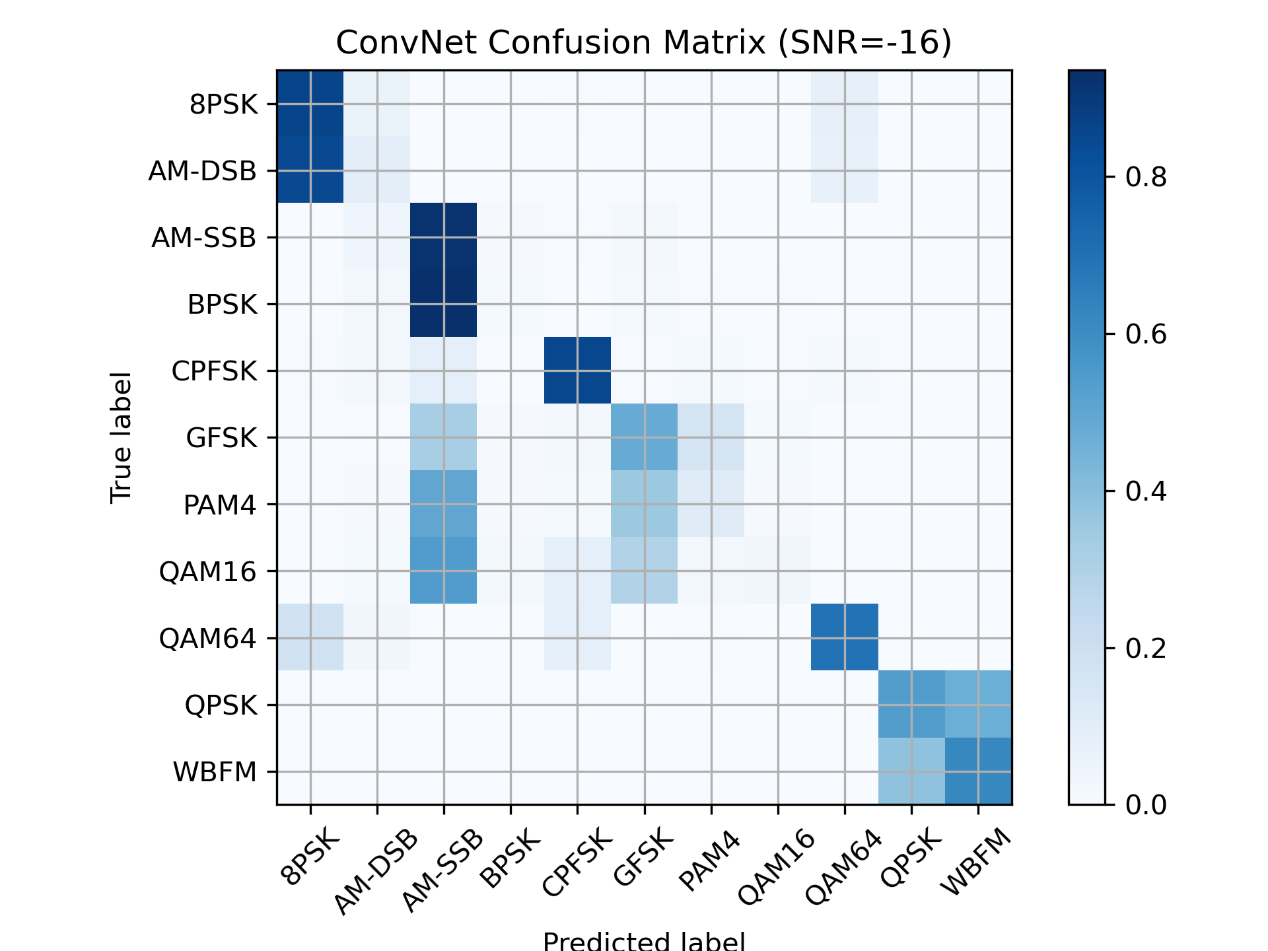
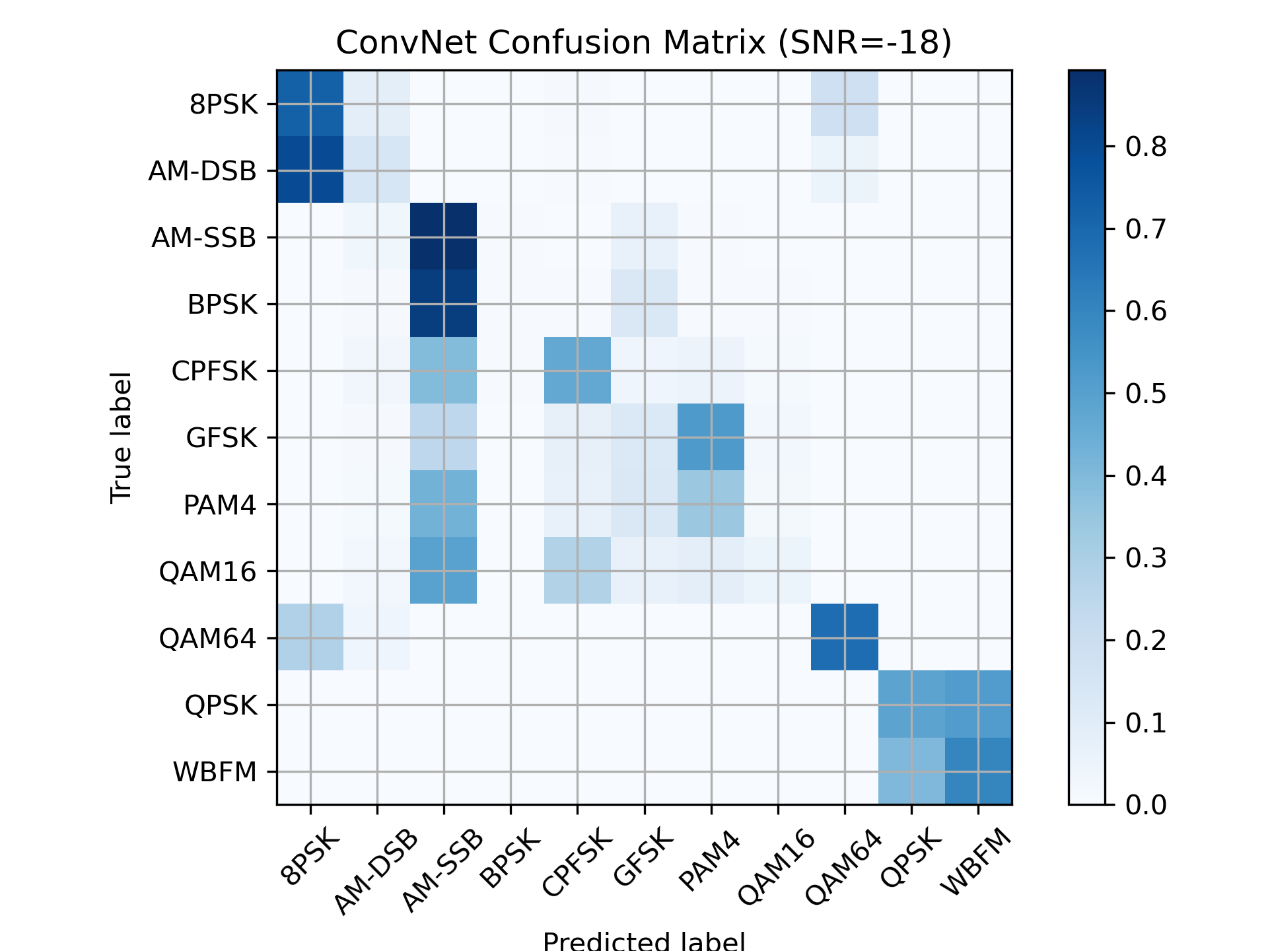
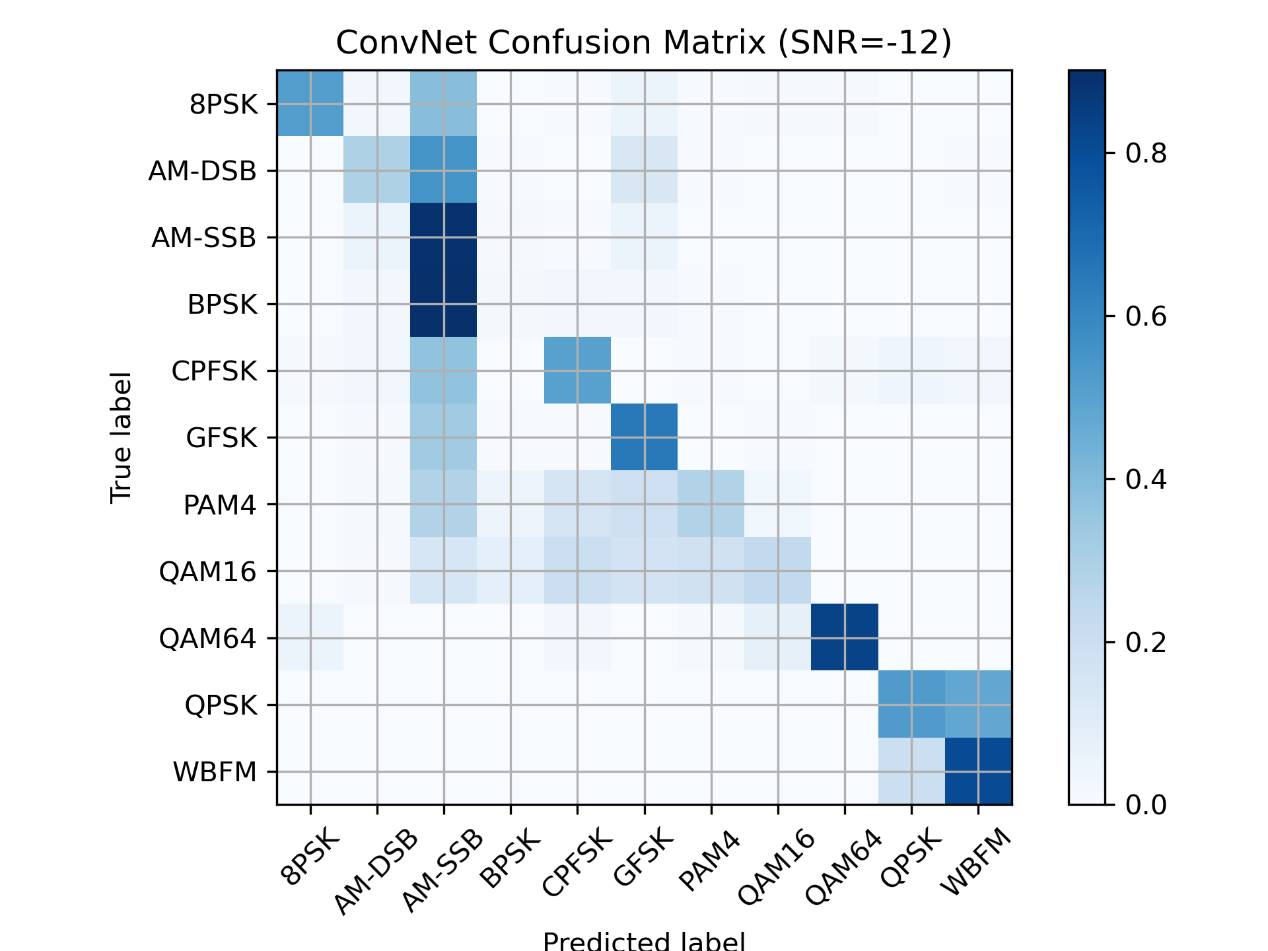
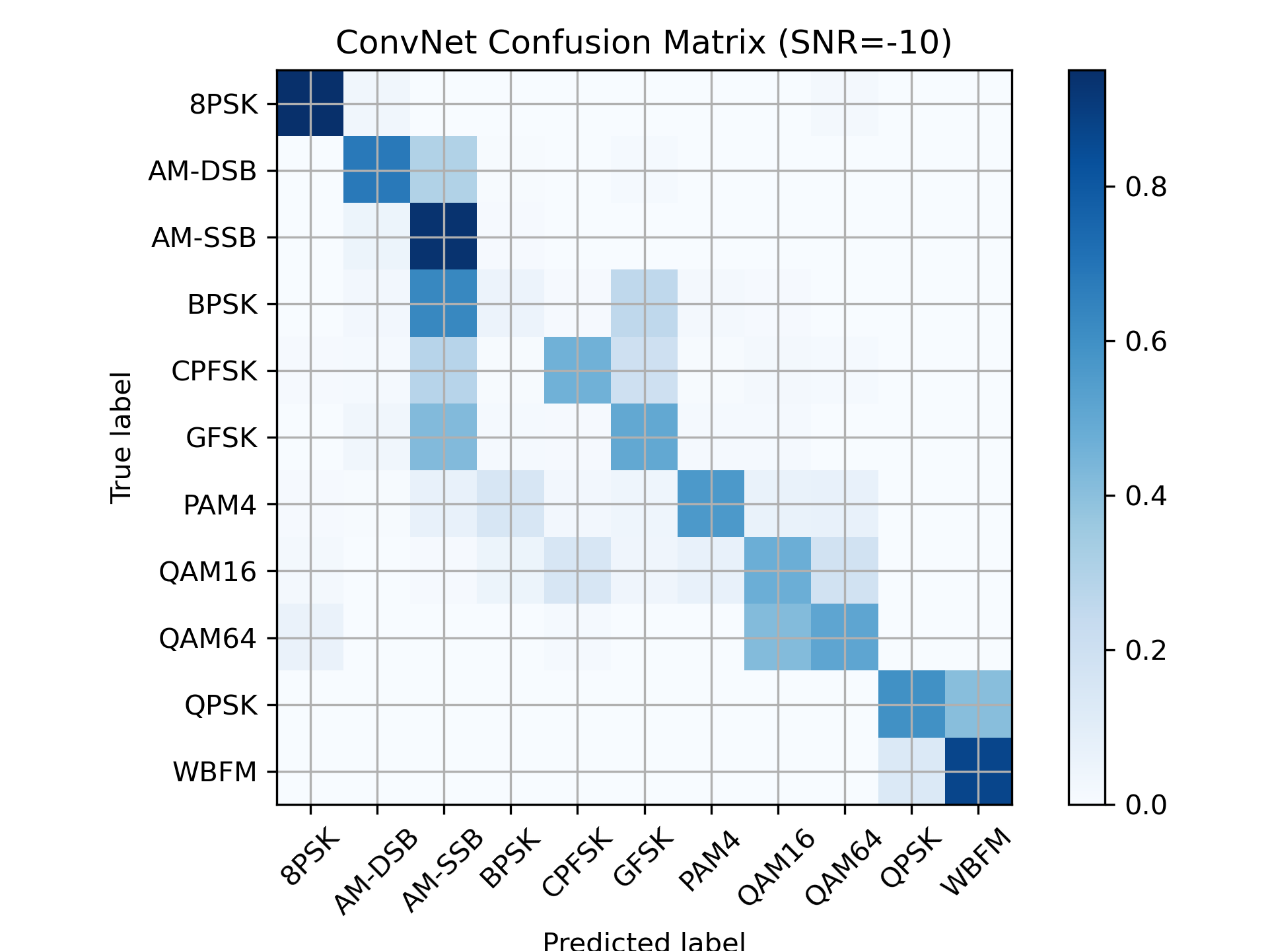


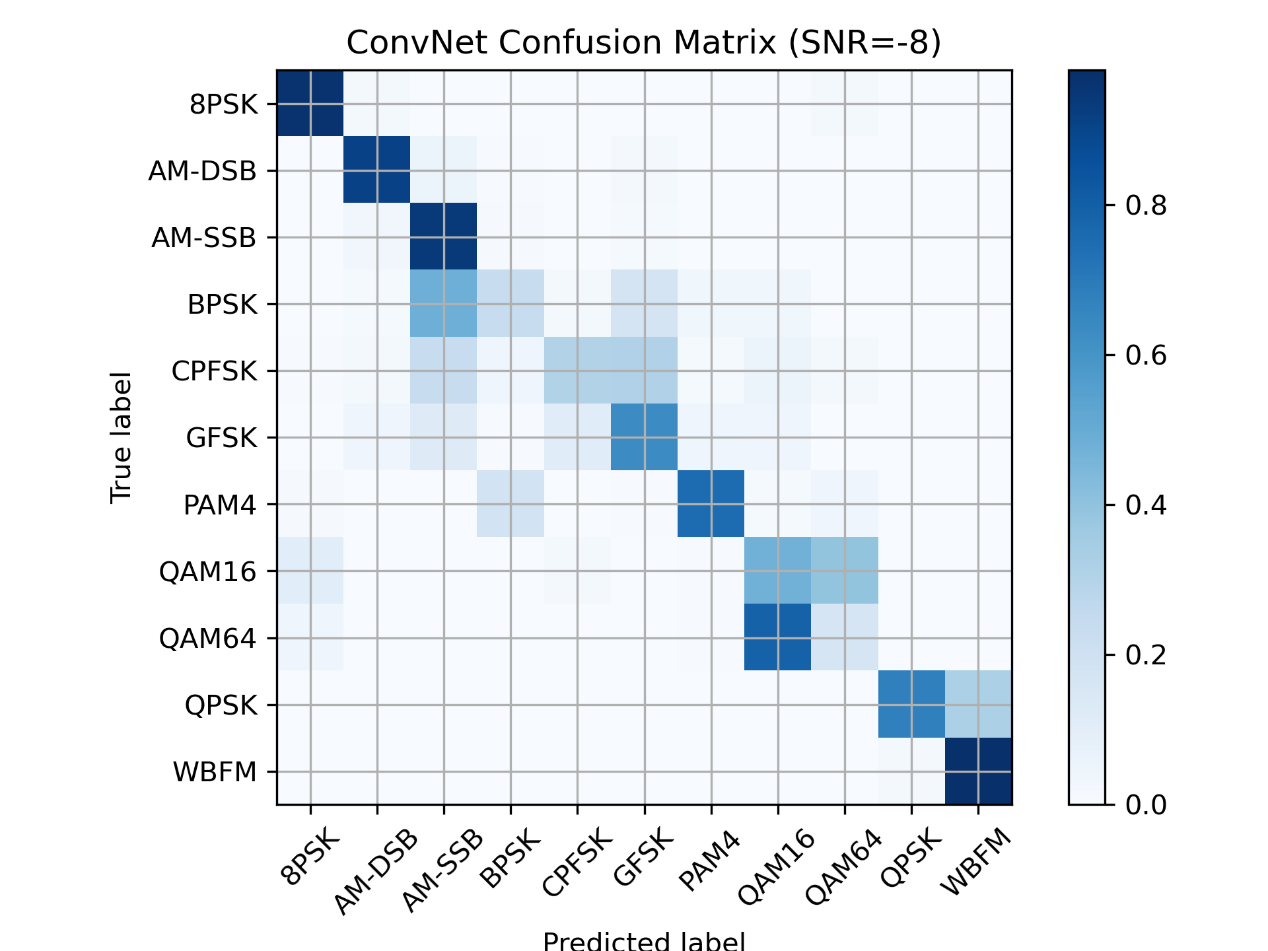
图3.3 整体混淆矩阵

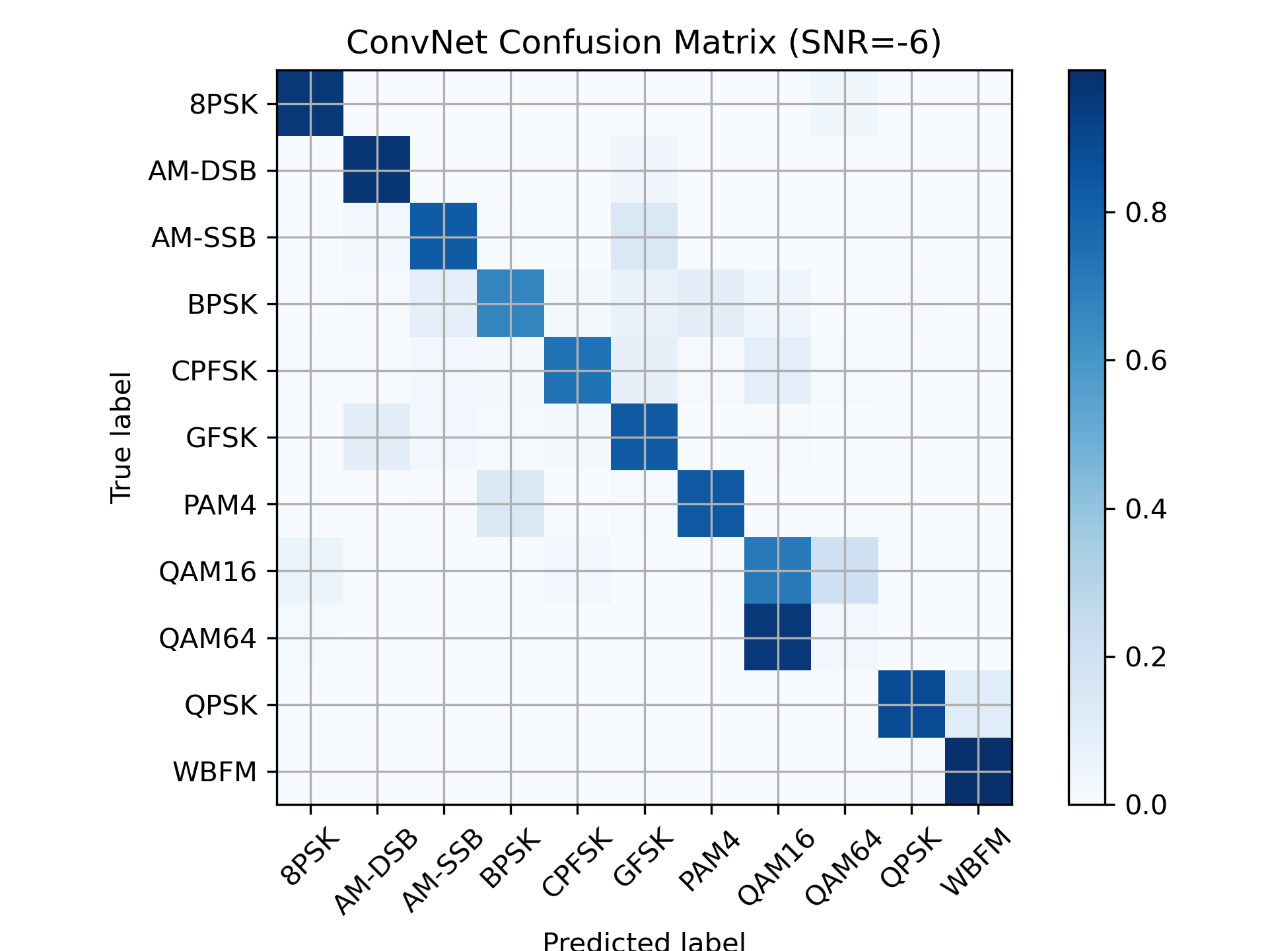


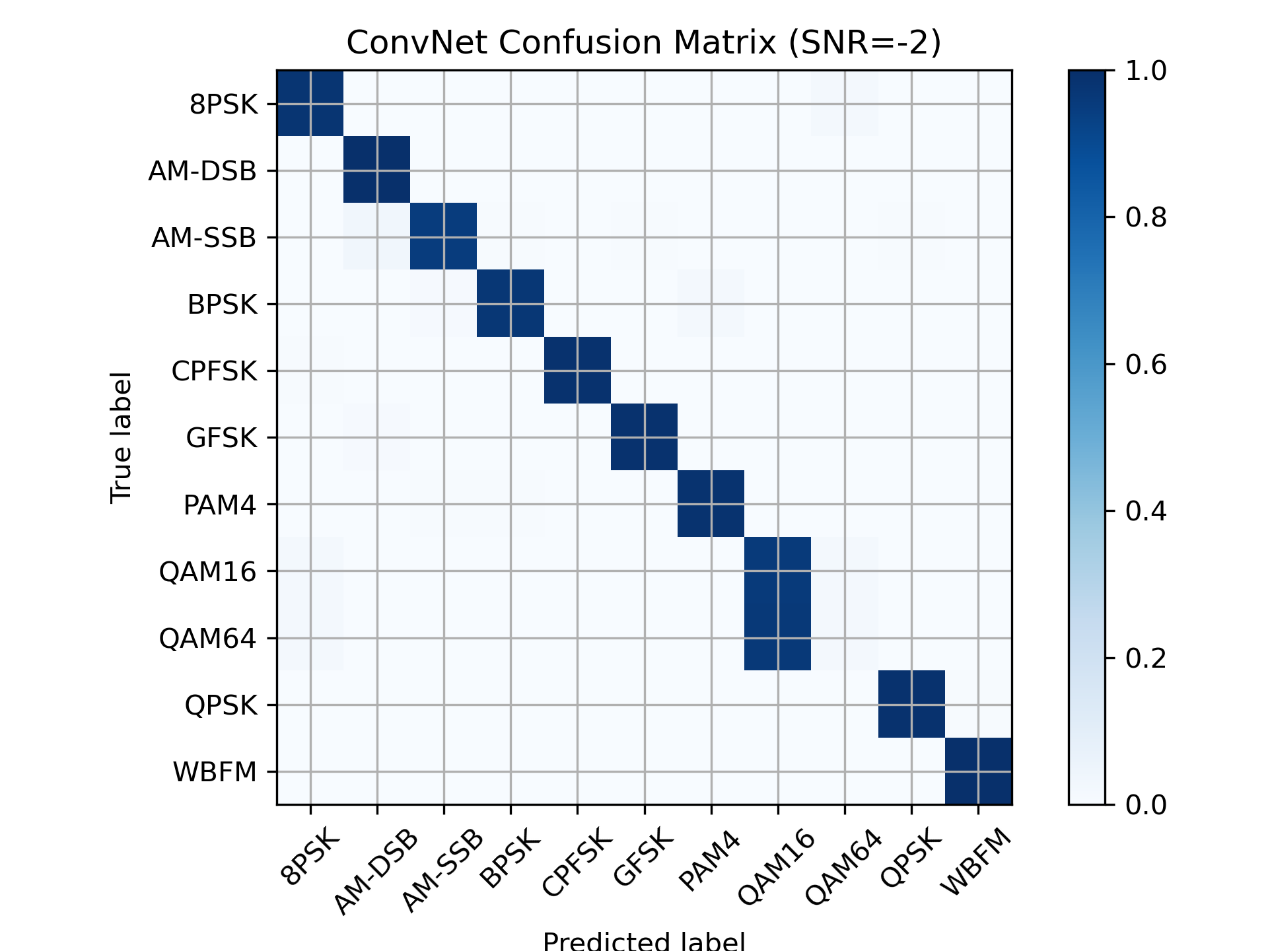
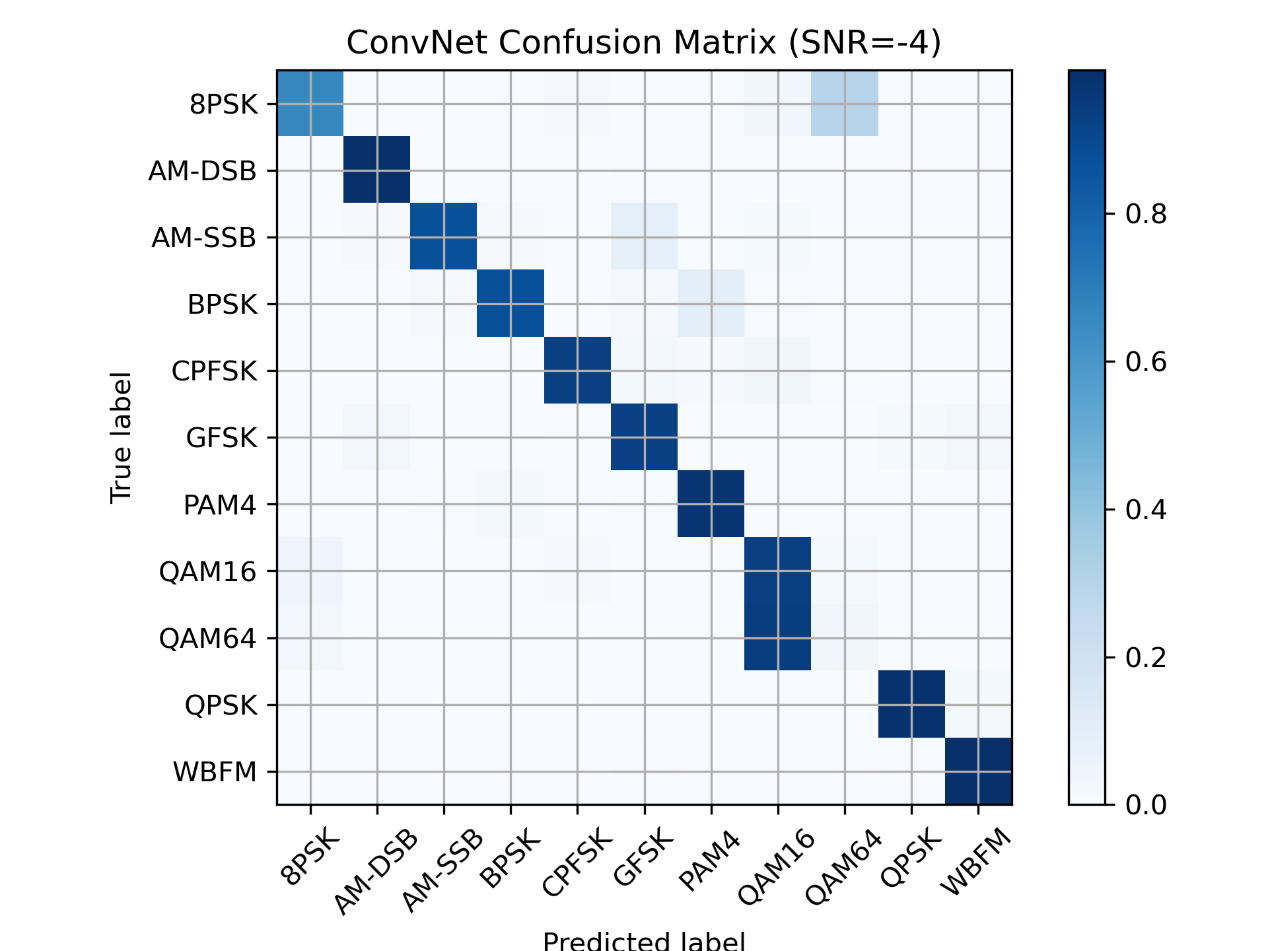


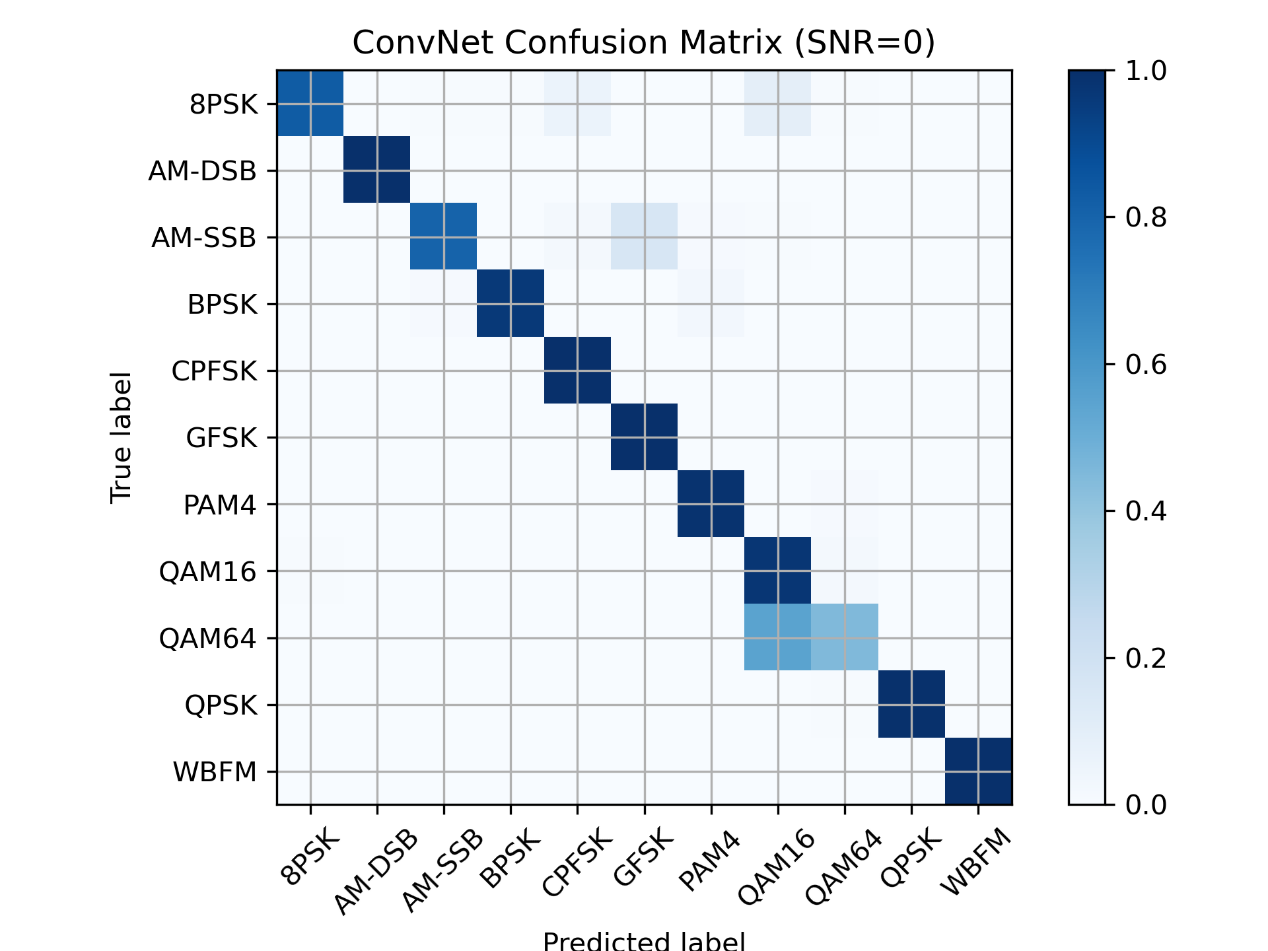


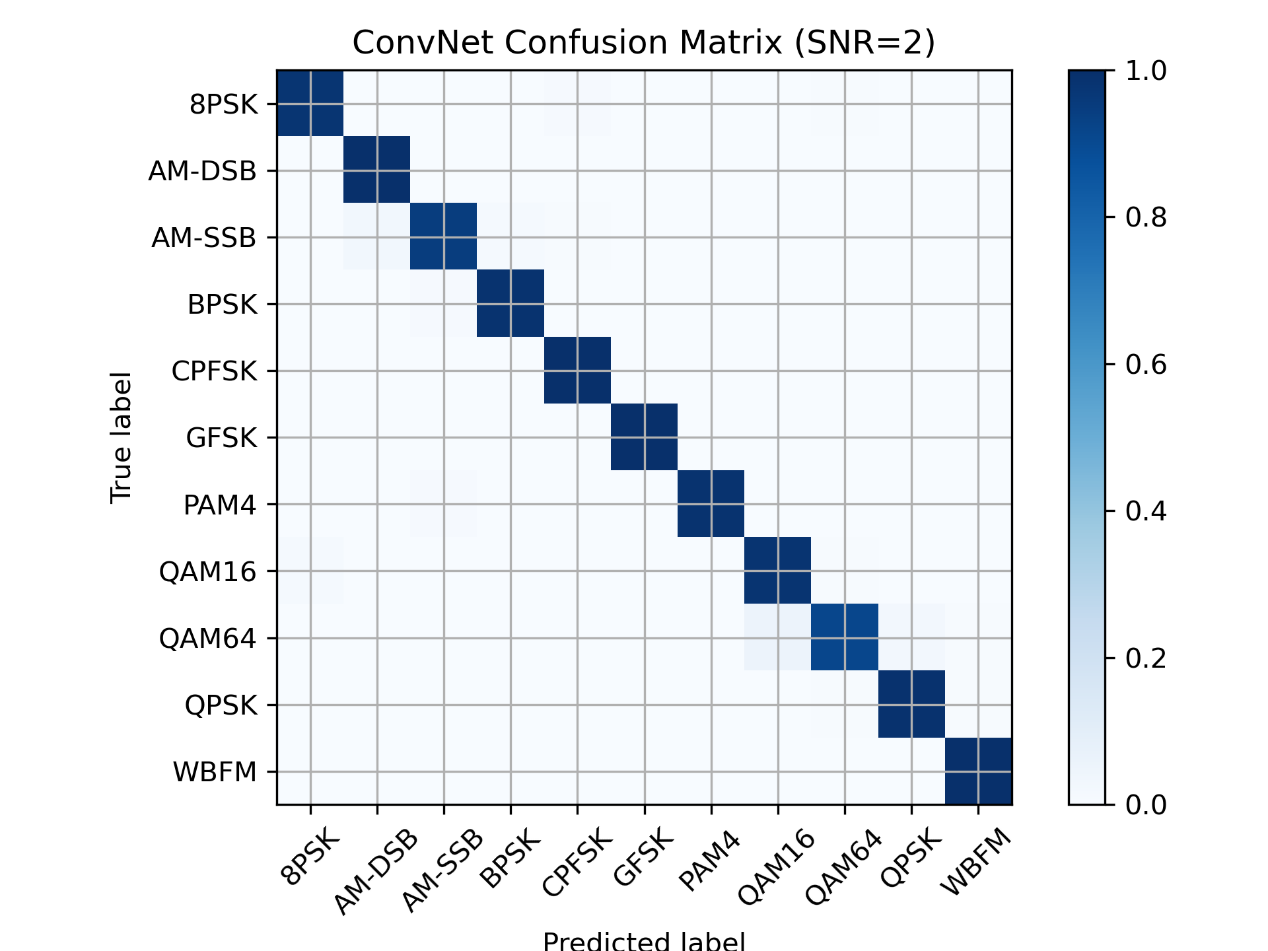


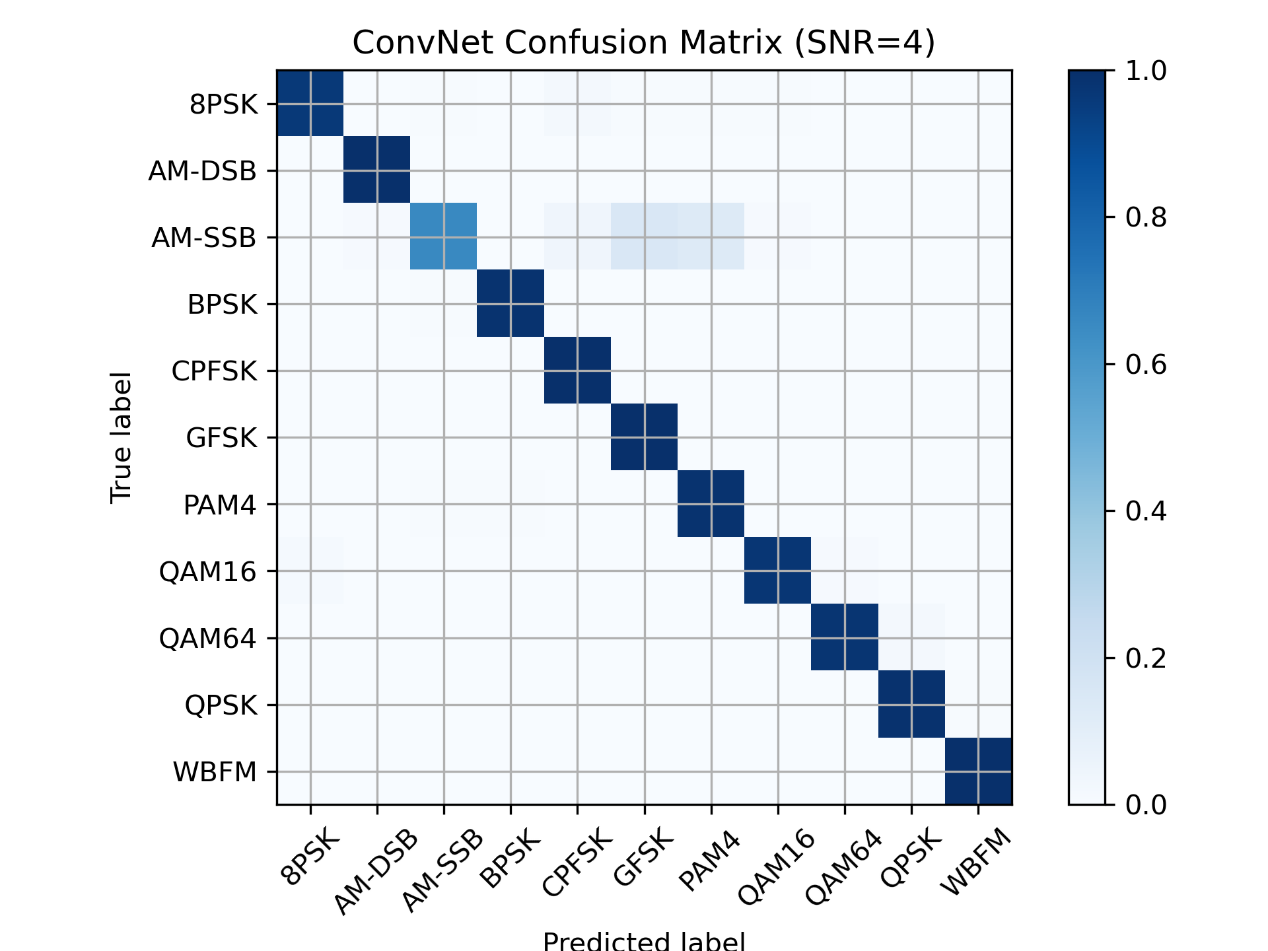


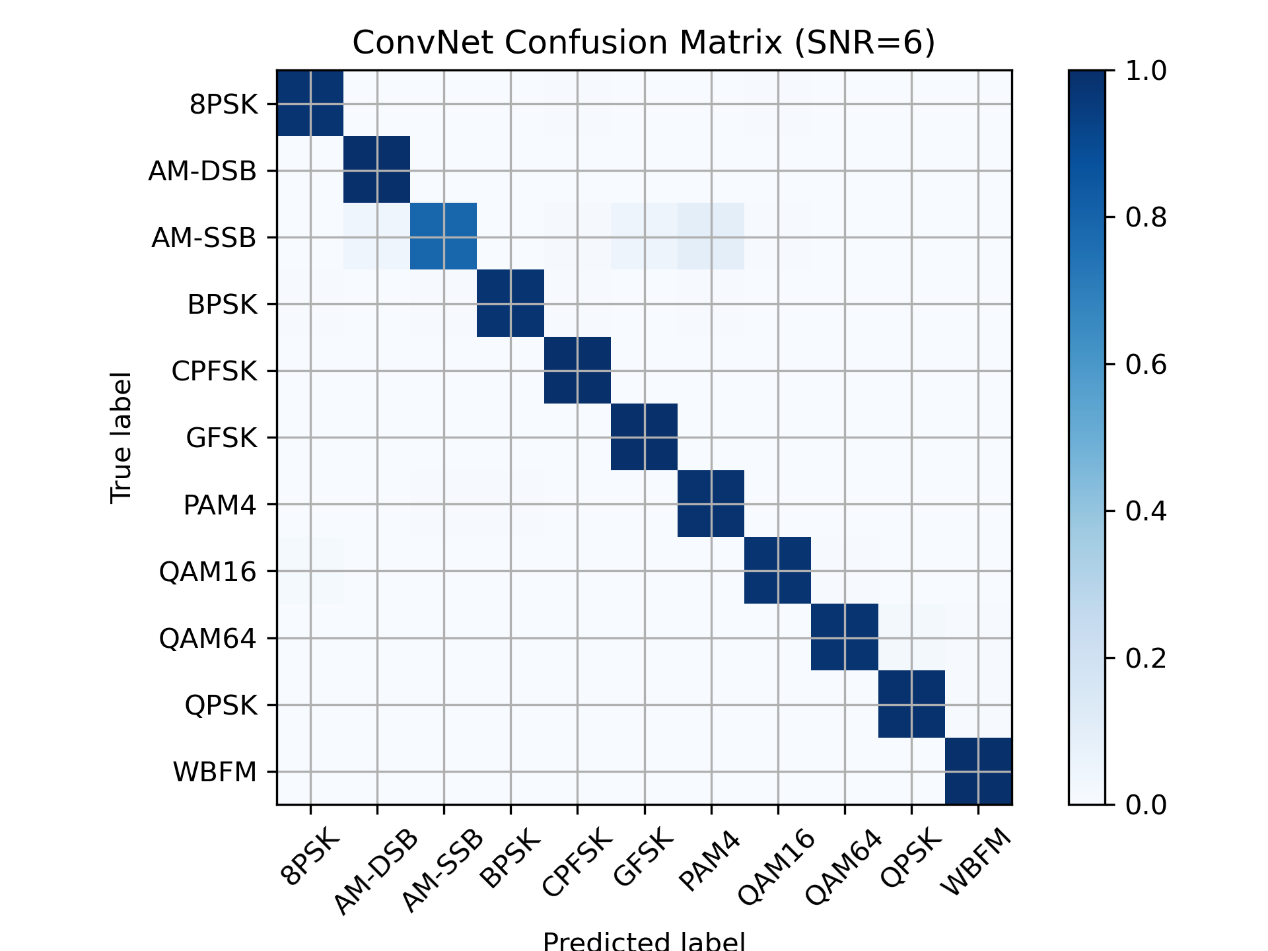


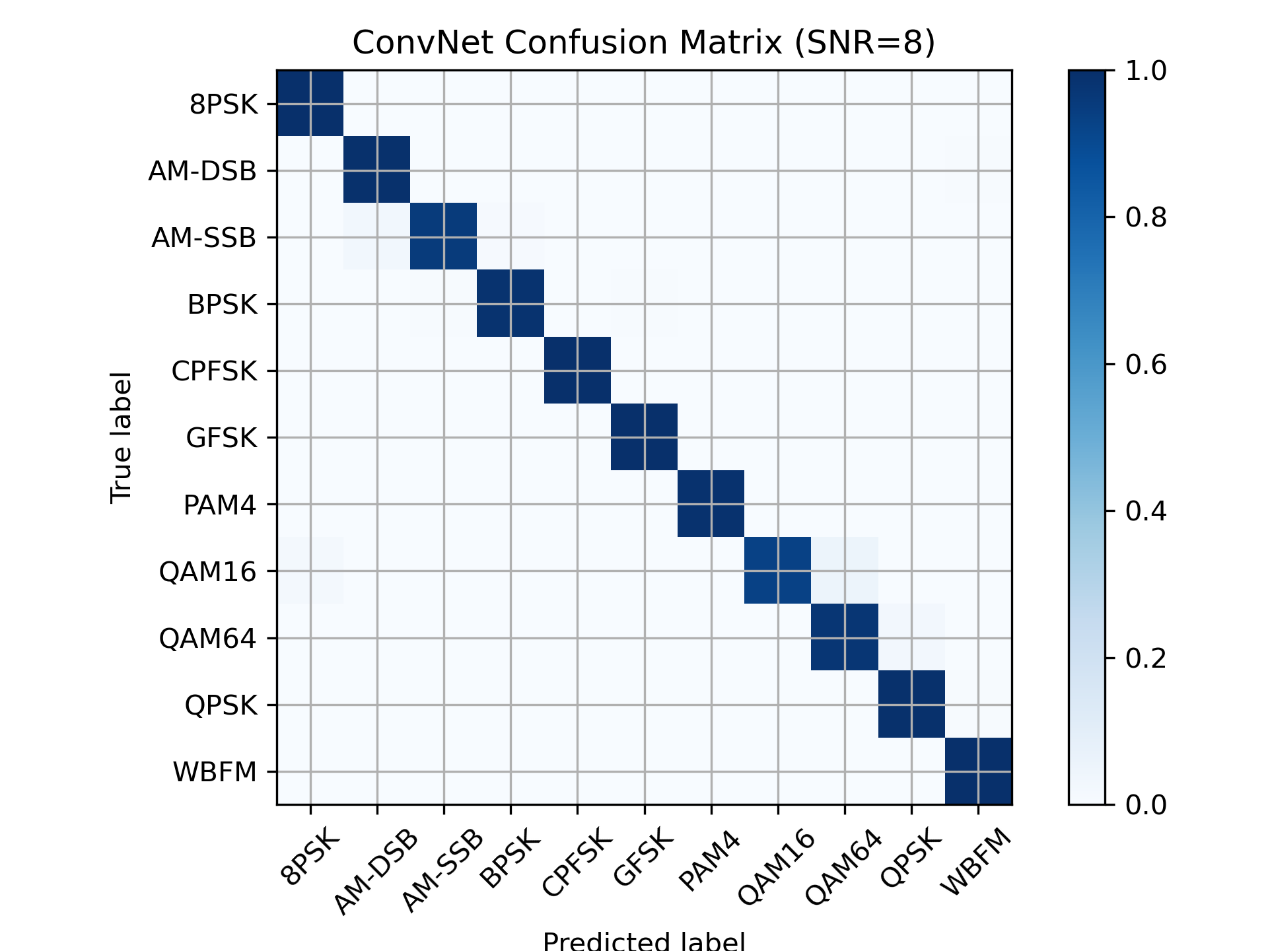


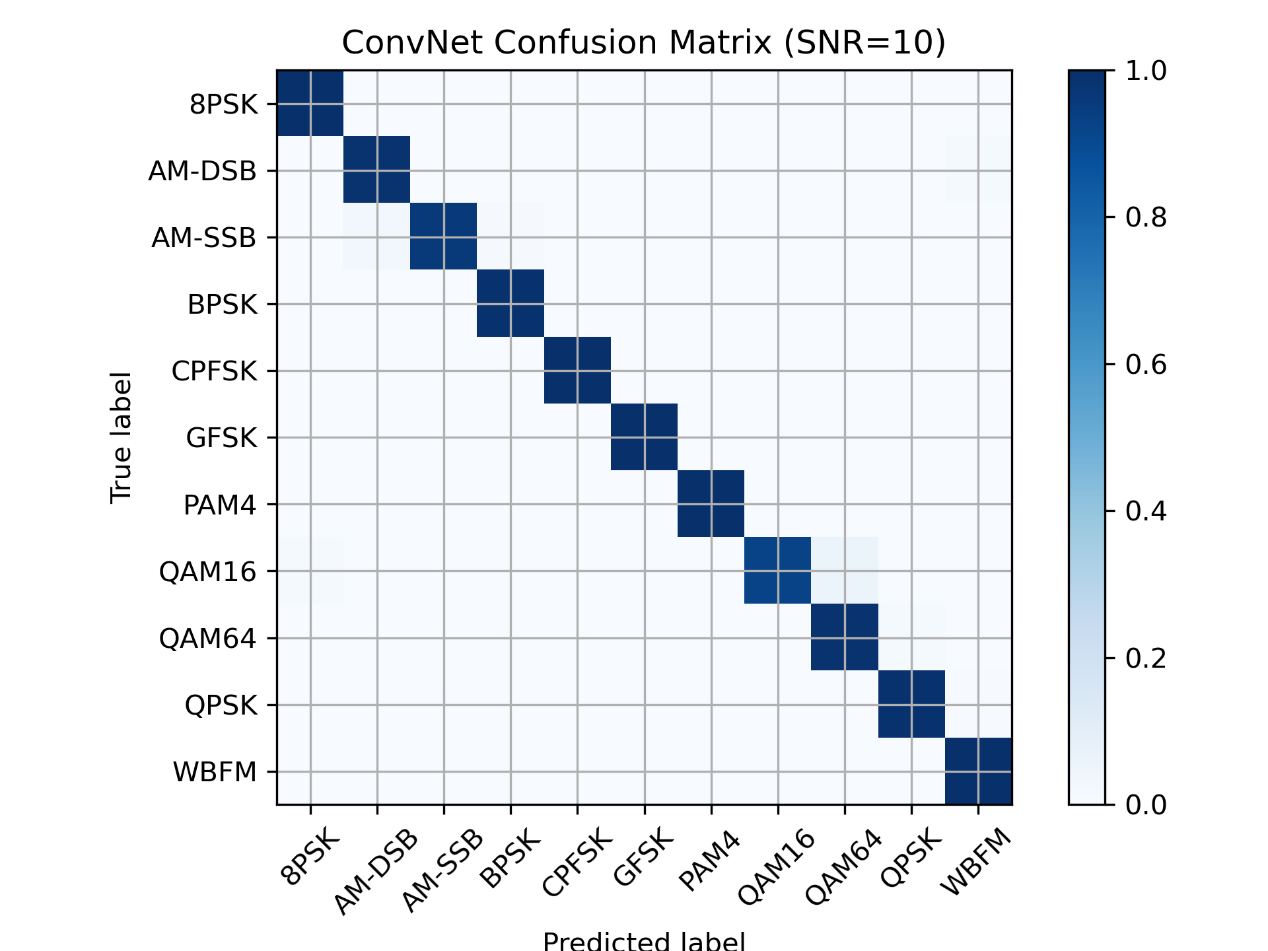


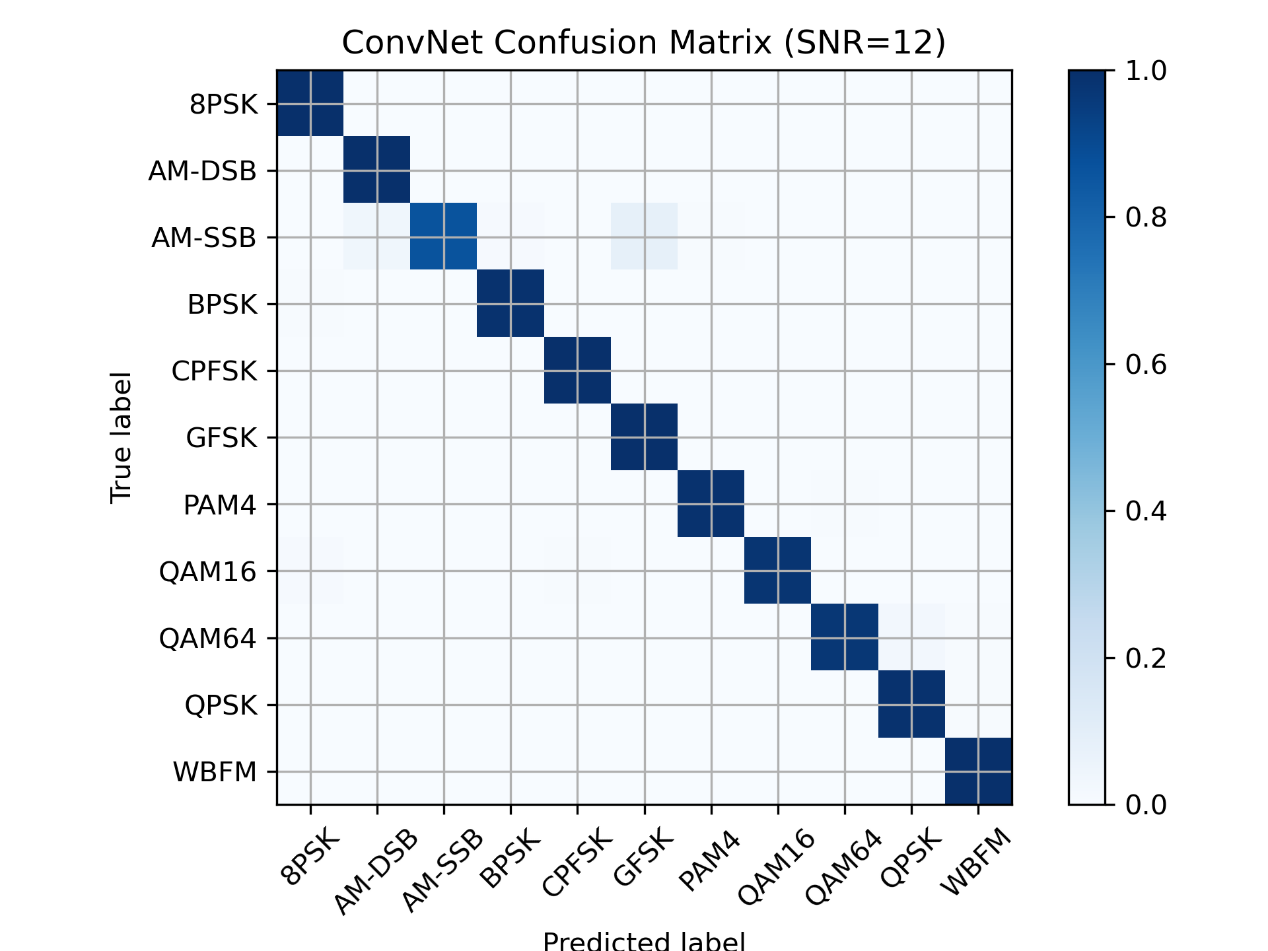


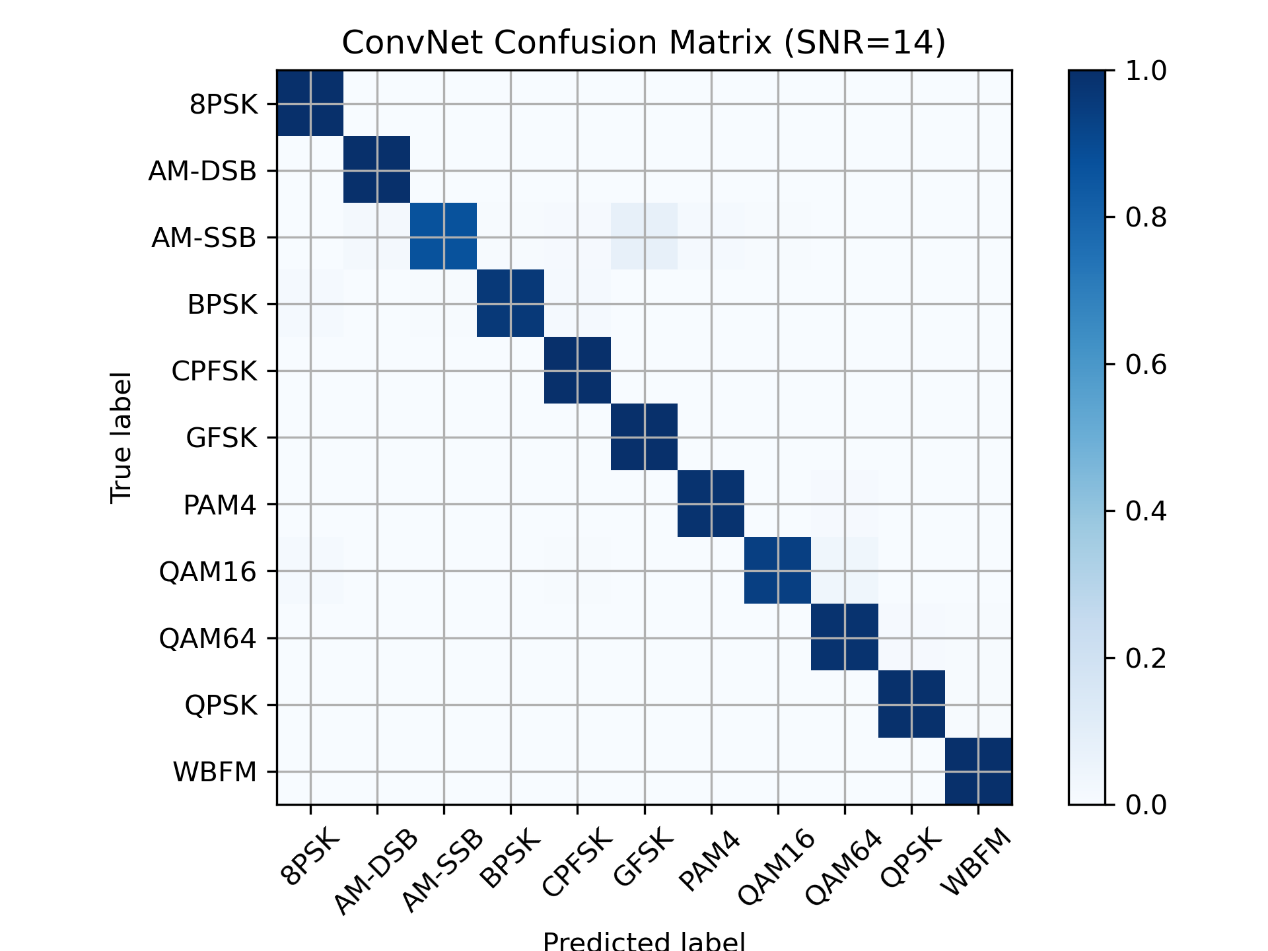


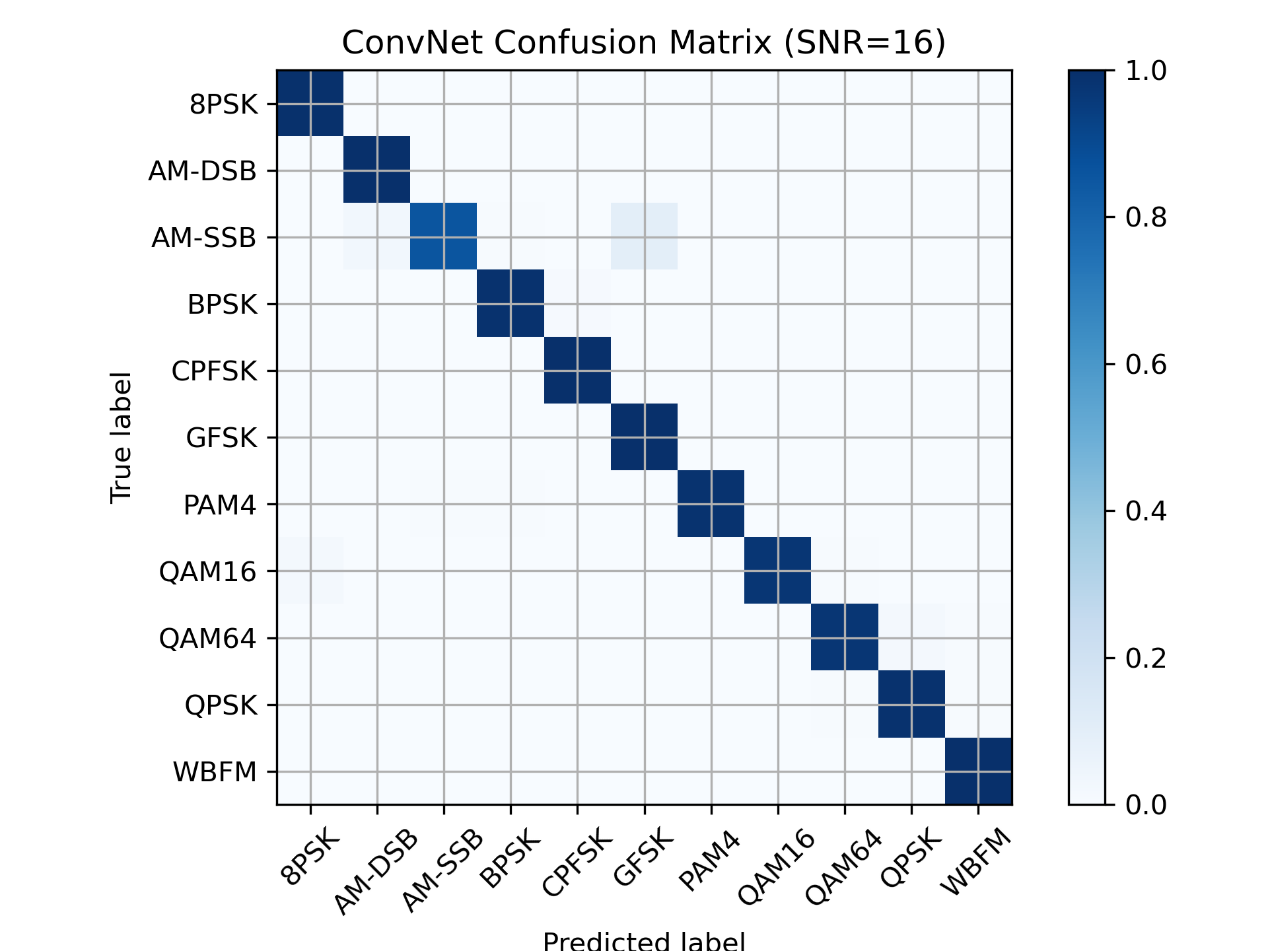












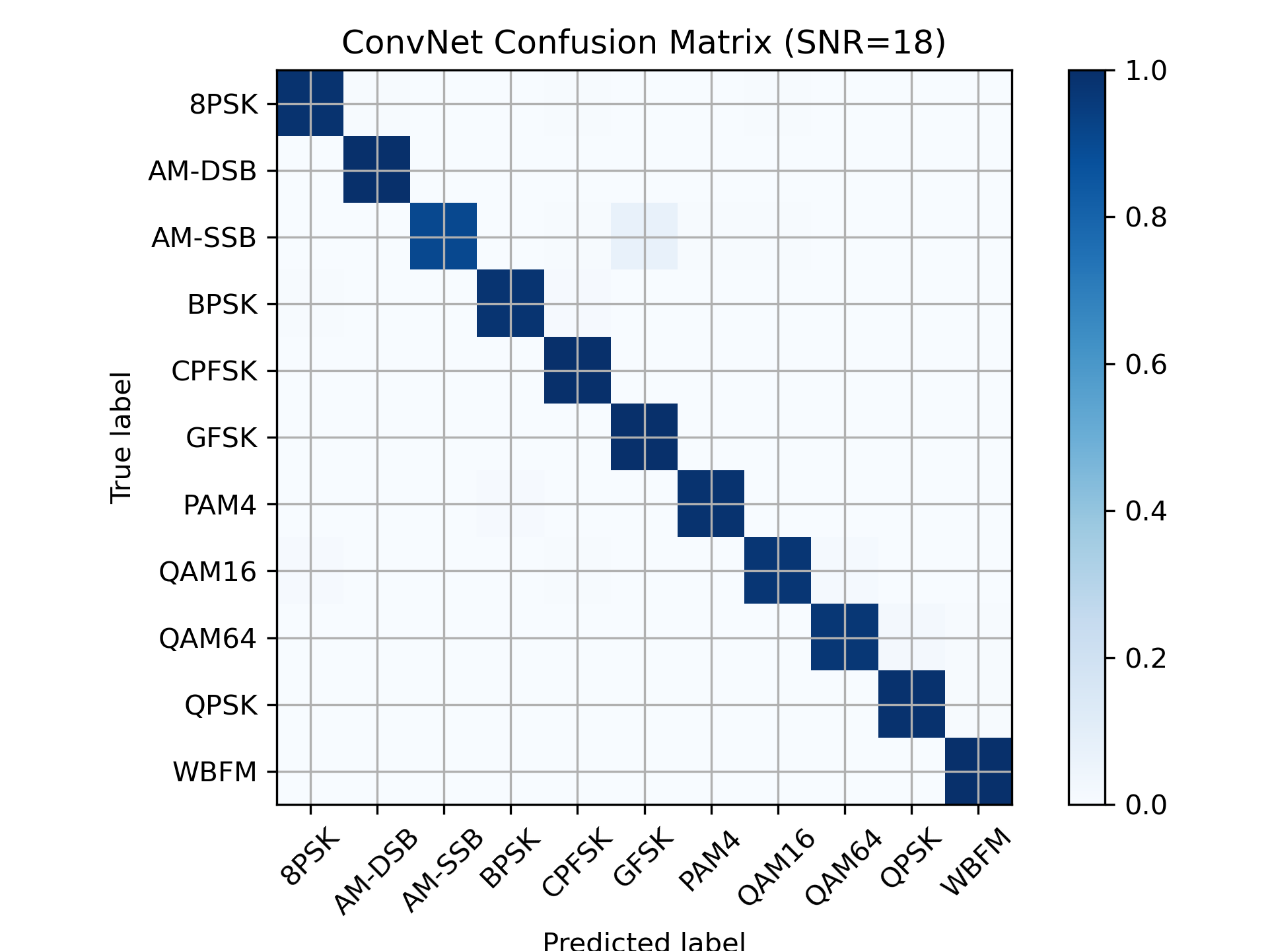


图3.4-3.24 各信噪比下混淆矩阵

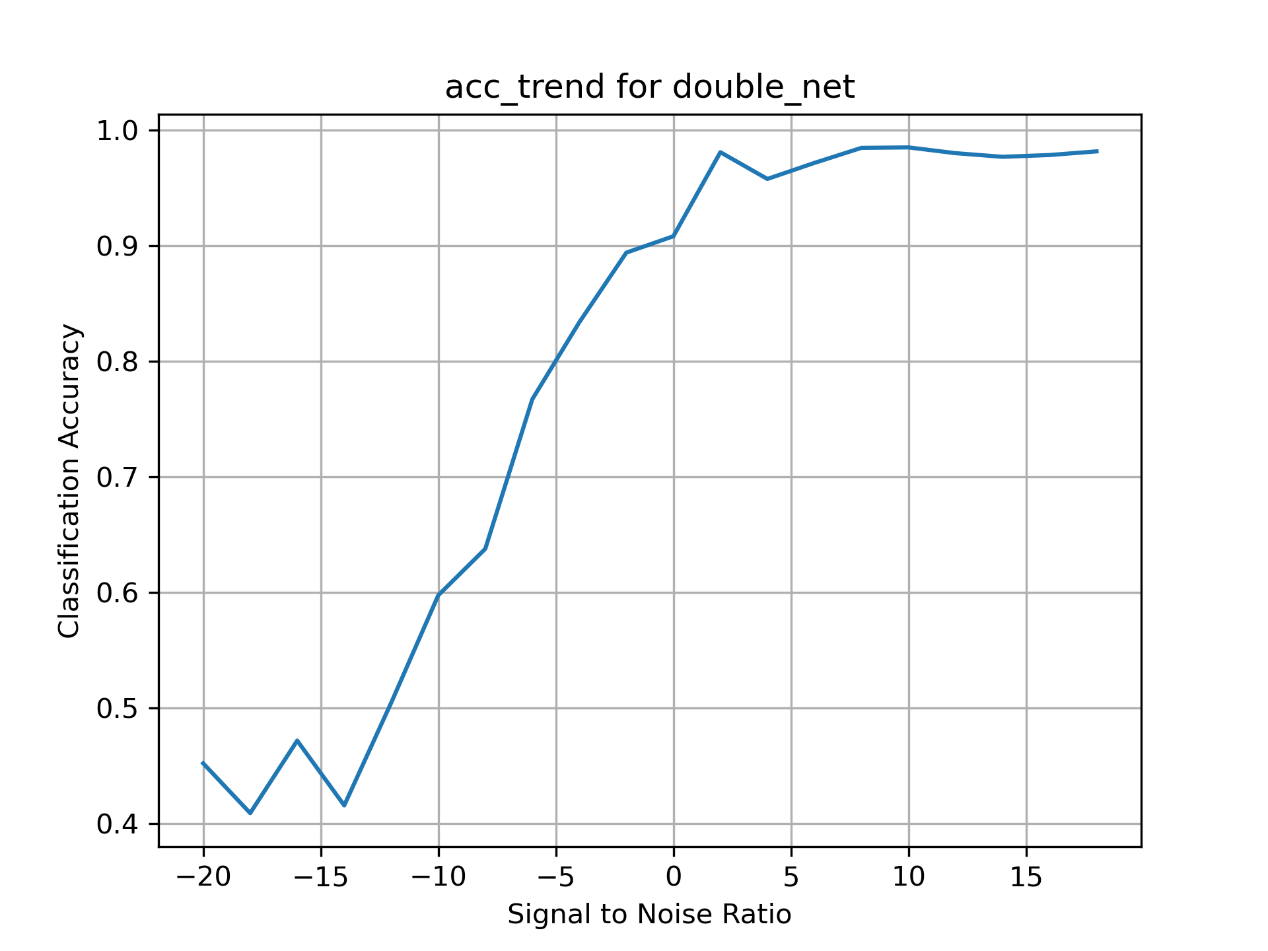


图3.25预测准确率随信噪比的变化曲线

# 四、总结分析

1.卷积神经网络所提取的自适应特征通常具有模糊性，致使特征冗余度无法消除，加入人工特征高阶累积量进行针对性得强化某些特征，会极大加快网络收敛速度，降低损失阈值，同时提高网络预测准确度。

2.本实验对当前调制识别研究现状下单一网络的不足之处进行了改进强化，使网络性能达到了前沿论文的指标，0db信噪比以上情况准确度达到90%，5db信噪比下预测准确度达到98%.该研究所提供的方法也对其他应用的深度学习研究有较大的借鉴意义。

# 五．附仿真核心代码、

try:

import tensorflow as tf

import os

os.environ["KERAS\_BACKEND"] = "tensorflow"

os.environ["THEANO\_FLAGS"] = "device=cuda,floatX=float32"

import numpy as np

import matplotlib.pyplot as plt

import pickle as cPickle

from scipy.stats import moment

from mix\_moment.mixd\_moment import mixed\_moment,self\_moments

ID = input("请告诉我你的学号后三位，我可以为你生成专属数据集！").strip()

print("Num GPUs Available: ", len(tf.config.experimental.list\_physical\_devices('GPU')))

# Load the dataset ...

# You will need to seperately download or generate this file

dataFile = "./module\_signal/RML2016.10a\_dict.pkl"

with open(dataFile, 'rb') as f:

Xd = cPickle.load(f, encoding='latin1')

snrs,mods = map(lambda j: sorted(list(set(map(lambda x: x[j], Xd.keys())))), [1,0])

# print('torch GPU:', torch.cuda.is\_available())

# print('tensorflow GPU:', tf.test.is\_gpu\_available())

# 以字典形式存储数据

X = []

lbl = []

for mod in mods:

for snr in snrs:

X.append(Xd[(mod,snr)][:600])

for i in range(600): lbl.append((mod,snr))

X = np.vstack(X)

# Partition the data

# into training and test sets of the form we can train/test on

# while keeping SNR and Mod labels handy for each

# 设置随机种子（确保每次运行代码选择的数据划分都是一样的）

np.random.seed(int(ID))

# 定义数据集大小

n\_examples = X.shape[0]

# 取一半的数据集作为训练集

n\_train = n\_examples \* 0.5

# 选择训练和测试索引

train\_idx = np.random.choice(range(0,n\_examples), size=int(n\_train), replace=False)

test\_idx = list(set(range(0,n\_examples))-set(train\_idx))

X\_train = X[train\_idx]

X\_test = X[test\_idx]

# 将标签转换为one-hot编码形式

def to\_onehot(yy):

yy1 = np.zeros([len(yy), max(yy)+1])

yy1[np.arange(len(yy)),yy] = 1

return yy1

Y\_train = to\_onehot(list(map(lambda x: mods.index(lbl[x][0]), train\_idx)))

Y\_test = to\_onehot(list(map(lambda x: mods.index(lbl[x][0]), test\_idx)))

# 从训练数据中提取应该输入的形状并打印出来

in\_shp = list(X\_train.shape[1:])

print(X\_train.shape, in\_shp)

classes = mods

#AP数据提取，作为人工特征

filename = './A\_P\_data.pickle'

with open(filename, 'rb') as file:

M = cPickle.load(file)

# 构建最大半径与最小半径之比为特征参数区分QAM族

Param\_R = np.zeros(n\_examples)

for i in range(n\_examples):

Param\_R[i] = np.max(M[i][0])/np.min(M[i][0])

# 以1到8阶矩作为特征参数

# 先计算出复随机变量的混合矩

comp\_data\_all = []

comp\_data = np.zeros(X[0].shape[1])

mom\_10 = []

mom\_20 = []

mom\_21 = []

mom\_30 = []

mom\_40 = []

mom\_41 = []

mom\_42 = []

mom\_50 = []

mom\_60 = []

mom\_63 = []

mom\_70 = []

mom\_80 = []

for i in range(n\_examples):

comp\_data = X[i][0] + 1j \* X[i][1]

mom\_10.append(self\_moments(comp\_data,[1]))

comp\_data\_all.append(comp\_data)

mom\_20.append(self\_moments(comp\_data,[2]))

mom\_21.append(mixed\_moment(comp\_data,np.conj(comp\_data),1,1))

mom\_30.append(self\_moments(comp\_data,[3]))

mom\_40.append(self\_moments(comp\_data,[4]))

mom\_41.append(mixed\_moment(comp\_data,np.conj(comp\_data),3,1))

mom\_42.append(mixed\_moment(comp\_data,np.conj(comp\_data),2,2))

mom\_50.append(self\_moments(comp\_data,[5]))

mom\_60.append(self\_moments(comp\_data,[6]))

mom\_63.append(mixed\_moment(comp\_data,np.conj(comp\_data),3,3))

mom\_70.append(self\_moments(comp\_data,[7]))

mom\_80.append(self\_moments(comp\_data,[8]))

# 以2到8阶累积量构造特征参数

# 计算高阶累积量，在这个列表推导式中，我们使用了 zip 函数，该函数用于将两个可迭代对象逐一配对，形成一个元组。

C\_20 = mom\_20

C\_21 = mom\_21

C\_40 = [mom\_40 - 3 \* mom\_20\*\*2 for mom\_40, mom\_20 in zip(mom\_40, mom\_20)]

C\_41 = [mom\_41 - 3 \* mom\_20\*mom\_21 for mom\_41, mom\_20,mom\_21 in zip(mom\_41, mom\_20,mom\_21)]

C\_42 = [mom\_42 - mom\_20\*\*2 - 2\*mom\_21\*\*2 for mom\_42, mom\_20, mom\_21 in zip(mom\_42, mom\_20, mom\_21)]

C\_60 = [mom\_60 - 15\*mom\_40\*mom\_20 + 30\*mom\_20\*\*3 for mom\_60, mom\_40, mom\_20 in zip(mom\_60, mom\_40, mom\_20)]

C\_63 = [mom\_63 - 9\*C\_42\*C\_21 - 6\*C\_21\*\*3 for mom\_63, C\_42, C\_21 in zip(mom\_63, C\_42, C\_21)]

C\_80 = [mom\_80 - 28\*mom\_60\*mom\_20 - 35\*mom\_40\*\*2 + 420\*mom\_40\*mom\_20\*\*2 - 630\*mom\_20\*\*4 for mom\_80, mom\_60,mom\_20,mom\_40

in zip(mom\_80, mom\_60,mom\_20,mom\_40)]

# 构建特征参数

# M\_1 = list(map(lambda x,y:np.abs(x/y), C\_20,C\_21))

# M\_2 = list(map(lambda x,y:np.abs(x/y\*\*2), C\_42,C\_21))

# M\_3 = list(map(lambda x,y:np.abs(x/y), C\_40,C\_42))

# M\_4 = list(map(lambda x,y:np.abs(x/y\*\*2), C\_40,C\_21))

M\_1 = [np.abs(c20 / c21) for c20, c21 in zip(C\_20, C\_21)]

M\_2 = [np.abs(c42 / c21\*\*2) for c42, c21 in zip(C\_42,C\_21)]

M\_3 = [np.abs(C\_40 / C\_42) for C\_40,C\_42 in zip(C\_40,C\_42)]

M\_4 = [np.abs(c40 / c21\*\*2) for c40, c21 in zip(C\_40,C\_21)]

M\_5 = [np.abs(c63 / c21\*\*3) for c63, c21 in zip(C\_63,C\_21)]

M\_6 = [np.abs(c80 / c21\*\*4) for c80, c21 in zip(C\_80,C\_21)]

extraData = [Param\_R,M\_1,M\_2,M\_3,M\_4,M\_5,M\_6,np.real(C\_60),np.imag(C\_60)]

extraData\_same = [np.array(item).flatten() for item in extraData]

# 将所有子数组堆叠为一个二维数组

extraData\_np = np.array(extraData\_same).reshape((len(extraData\_same[0]),len(extraData\_same)))

extraData\_train = np.real(extraData\_np[train\_idx])

extraData\_test = np.real(extraData\_np[test\_idx])

# 将 NumPy 数组转换为 TensorFlow 张量

extraData\_train\_tf = tf.convert\_to\_tensor(extraData\_train, dtype=tf.float32)

extraData\_test\_tf = tf.convert\_to\_tensor(extraData\_test, dtype=tf.float32)

in\_shp\_2 = list(extraData\_train.shape[1:])

print(extraData\_train.shape,in\_shp\_2)

extra\_Param = tf.keras.layers.Input(shape=(in\_shp\_2))

#第二层

dr = 0.5

input\_2 = tf.keras.layers.Input(shape=in\_shp)

b = tf.keras.layers.Reshape(in\_shp+[1])(input\_2)

b = tf.keras.layers.ZeroPadding2D((0, 2), data\_format="channels\_last")(b)

b = tf.keras.layers.Convolution2D(256, (1, 3),padding='valid', activation="relu", name="conv4", kernel\_initializer='glorot\_uniform', data\_format="channels\_last")(b)

b = tf.keras.layers.BatchNormalization(axis=1,momentum=0.99,epsilon=0.001)(b)

b = tf.keras.layers.MaxPooling2D(pool\_size=(1, 2), strides=None, padding='valid', data\_format=None)(b)

b = tf.keras.layers.ZeroPadding2D((0, 2), data\_format="channels\_last")(b)

b = tf.keras.layers.Convolution2D(80, (2, 3), padding='valid', activation="relu", name="conv5", kernel\_initializer='glorot\_uniform', data\_format="channels\_last")(b)

b = tf.keras.layers.BatchNormalization(axis=1,momentum=0.99,epsilon=0.001)(b)

b = tf.keras.layers.MaxPooling2D(pool\_size=(1, 2), strides=None, padding='valid', data\_format=None)(b)

b = tf.keras.layers.ZeroPadding2D((0, 2), data\_format="channels\_last")(b)

b = tf.keras.layers.Convolution2D(80, (1, 3), padding='valid', activation="relu", name="conv6", kernel\_initializer='glorot\_uniform', data\_format="channels\_last")(b)

b = tf.keras.layers.BatchNormalization(axis=1,momentum=0.99,epsilon=0.001)(b)

b = tf.keras.layers.MaxPooling2D(pool\_size=(1, 2), strides=None, padding='valid', data\_format=None)(b)

b = tf.keras.layers.Reshape((17, 80),input\_shape=(1, 17, 80))(b)

b = tf.keras.layers.LSTM(100,activation='relu', recurrent\_activation='hard\_sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', recurrent\_initializer='orthogonal', return\_sequences=True)(b)

b = tf.keras.layers.LSTM(50, activation='relu', recurrent\_activation='hard\_sigmoid', use\_bias=True, kernel\_initializer='glorot\_uniform', recurrent\_initializer='orthogonal', return\_sequences=True)(b)

output\_2 = tf.keras.layers.Flatten()(b)

second = tf.keras.Model(input\_2, output\_2, name="second")

second.summary()

#两个网络相连接

concate = tf.keras.layers.Concatenate()([extra\_Param, output\_2])

print(concate.shape)

concate = tf.keras.layers.Dense(256, activation='relu', kernel\_initializer='he\_normal', name="dense")(concate)

concate = tf.keras.layers.Dropout(dr)(concate)

print(concate.shape)

concate = tf.keras.layers.Dense(len(classes), kernel\_initializer='he\_normal', name='dense2')(concate)

concate = tf.keras.layers.Activation('softmax')(concate)

print(concate.shape)

output = tf.keras.layers.Reshape([len(classes)])(concate)

final = tf.keras.Model(inputs=[extra\_Param, input\_2], outputs=output)

final.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

final.summary()

# 设置一些参数

nb\_epoch = 100 # number of epochs to train on要训练的历元数

batch\_size = 512 # training batch size训练批量大小

# perform training ...

# - call the main training loop in keras for our network+dataset调用keras中主训练循环

# filepath1 = 'convmodrecnets1\_CNN2\_0.5.wts.h5'

# filepath2 = 'convmodrecnets2\_CNN2\_0.5.wts.h5'

filepath3 = './MyNet.wts.h5'

# Set up some params

history3 = final.fit([extraData\_train\_tf, X\_train], #训练数据

Y\_train, #训练数据对应的标签

batch\_size=batch\_size, #训练批量大小，每次训练时使用的样本数

epochs=nb\_epoch, #训练轮数，表示模型需要训练的次数

#show\_accuracy=False, #表示不显示训练过程中的准确率

verbose=2, #表示在训练过程中显示详细信息

validation\_data=([extraData\_test\_tf, X\_test],Y\_test), #使用测试数据X\_test和标签Y\_text进行模型的验证

callbacks = [ #回调函数，用于在每个训练轮数结束时保存模型的权重

tf.keras.callbacks.ModelCheckpoint(filepath3, monitor='val\_loss', verbose=0, save\_best\_only=True, mode='auto'),#filepath是保存模型权重的路径

tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5, verbose=0, mode='auto')#回调函数，用于在验证集上损失函数不再下降时停止训练，5次迭代都没有改进时停止训练

])

print ("训练已经完成，最佳权重模型已保存到根目录！正在加载最佳权重模型....")

final.load\_weights(filepath3)

print ("最佳模型加载成功！")

score = final.evaluate([extraData\_test\_tf, X\_test], Y\_test, verbose=0, batch\_size=batch\_size)

print("Loss: ", score[0])

print("Accuracy: ", score[1])

# with open('history.json', 'w') as f:

# json.dump(history.history, f)

# json.dump(history.epoch, f)

#

# 画出损失曲线

plt.figure()

plt.title('Training performance')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.plot(history3.epoch, history3.history['loss'], label='train loss+error')

plt.plot(history3.epoch, history3.history['val\_loss'], label='val\_error')

plt.grid(True)

plt.legend()

plt.savefig("./loss", format='png', dpi=100)

# 定义混淆矩阵，输入为矩阵数据，标题，配色和标签，实际调用时只需要提供混淆矩阵数据和标签列表即可。

def plot\_confusion\_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues, labels=[]):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.grid(True)

# 批处理对测试集进行预测

test\_Y\_hat = final.predict([extraData\_test\_tf,X\_test], batch\_size=batch\_size)

conf = np.zeros([len(classes),len(classes)])

confnorm = np.zeros([len(classes),len(classes)])

# 遍历所有测试样本

for i in range(0,X\_test.shape[0]):

# 找到第i个测试样本的标签，因为是二进制编码格式。所以找到的是1的索引

j = list(Y\_test[i,:]).index(1)

# 找到预测结果中概率最大的元素索引

k = int(np.argmax(test\_Y\_hat[i,:]))

# 对应混淆矩阵上的点加1

conf[j,k] = conf[j,k] + 1

# 遍历所有类别

for i in range(0,len(classes)):

# 对原始混淆矩阵进行归一化

confnorm[i,:] = conf[i,:] / np.sum(conf[i,:])

plot\_confusion\_matrix(confnorm, labels=classes)

plt.savefig("./confu\_matrix\_total",format='png', dpi=100)

acc = {}

# 取出测试集的所有信噪比列表，根据测试集索引test\_idx来取

test\_SNRs = list(map(lambda x: lbl[x][1], test\_idx))

# 把指定信噪比下的测试数据提取出来

for snr in snrs:

# 作判断是不是指定信噪比

snr\_bool = np.array(test\_SNRs) == snr

# 找到匹配的信噪比索引

snr\_idx = np.where(snr\_bool)

# 取出该信噪比下的数据

test\_X\_i = X\_test[snr\_idx]

test\_M\_i = extraData\_test[snr\_idx]

test\_M\_i = tf.convert\_to\_tensor(test\_M\_i, dtype=tf.float32)

# test\_M\_i = np.reshape(test\_M\_i, (5456, 5))

# 取出标签

test\_Y\_i = Y\_test[snr\_idx]

print(len(snr\_idx[0]))

# estimate classes对该信噪比的测试集数据进行预测

test\_Y\_i\_hat = final.predict([test\_M\_i,test\_X\_i], batch\_size=batch\_size)

# 初始化混淆矩阵

conf1 = np.zeros([len(classes), len(classes)])

confnorm1 = np.zeros([len(classes), len(classes)])

# 遍历测试样本，构建原始混淆矩阵

for i in range(0,test\_X\_i.shape[0]):

j = list(test\_Y\_i[i,:]).index(1)

k = int(np.argmax(test\_Y\_i\_hat[i,:]))

conf1[j,k] = conf1[j,k] + 1

# 归一化混淆矩阵

for i in range(0,len(classes)):

confnorm1[i,:] = conf1[i,:] / np.sum(conf1[i,:])

plt.figure()

plot\_confusion\_matrix(confnorm1, labels=classes, title="ConvNet Confusion Matrix (SNR=%d)"%(snr))

plt.savefig("./comf\_Matrix\_for\_snr=" + str(snr), format='png', dpi=100) # 设置 dpi 参数以调整保存的图像质量

# 拿到原始混淆矩阵对角线的元素并求和

cor = np.sum(np.diag(conf1))

# 求出除了对角线元素外的所有元素的和

ncor = np.sum(conf1) - cor

# 总体准确率为预测对的数量比上总数

print("Overall Accuracy: ", cor / (cor+ncor))

acc[snr] = 1.0\*cor/(cor+ncor)

# Save results to a pickle file for plotting later

print(acc)

fd = open('./acc\_trend.dat', 'wb')

cPickle.dump( ("CNN2", 0.5, acc) , fd )

# Plot accuracy curve

plt.figure()

plt.plot(snrs, list(map(lambda x: acc[x], snrs)))

plt.xlabel("Signal to Noise Ratio")

plt.ylabel("Classification Accuracy")

plt.title("acc\_trend for double\_net")

plt.grid(True)

plt.savefig("./acc\_trend", format='png', dpi=100) # 设置 dpi 参数以调整保存的图像质量

except Exception as e:

input(e)

input('请按规定格式重新输入并确认')