Spectrum Sensing in Cognitive Radio using DetectNet

Report submitted to the SASTRA Deemed to be University

As the requirements for the course

CSE300 / INT300 / ICT300 - MINI PROJECT

Submitted by

Bala Shanmugam M
(Reg. No.: 124156028, CSE AI&DS)
Shyam Kumar Reddy K
(Reg. No.: 124156048, CSE AI&DS)

May 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA - 613401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Spectrum Sensing in Cognitive Radio using DetectNet"** submitted as a requirement for the course, CSE300 / INT300 / ICT300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Bala Shanmugam M (124156028, CSE AI&DS) and Mr. Shyam Kumar Reddy K (124156048, CSE AI&DS)** during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation :

Date :

Mini Project *Viva voce* held on ________________

Examiner 1 **Examiner 2**

Acknowledgments

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S.Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. S. Gopalakrishnan**, Associate Dean, Department of Computer Application, **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology

Our guide **Dr. Kannan. K**, Professor, School of Computing was the driving force
behind this whole idea from the start. His deep insight into the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through the project.

List of Figures

List of Tables

List of Abbreviations

CNN - Convolutional Neural Network

DNN - Deep Neural Network

LSTM - Long Short-Term Memory

Pd - Probability of detection

Pf - Probability of false alarm

SNR - signal-to-noise ratio

Pmd - Probability of missed detection

CSCG – Circularly Symmetric Complex Guassian

NOTATIONS

H0 – Null Hypothesis

H1 – Alternate Hypothesis

w(n) – White Noise

hs(n) – Occupied Signal

Abstract

Detection of primary user's signal for the secondary users to use idle licenced spectrum is essential. Energy detector is a conventional tool for the detection but it lacks the accuracy due to SNR wall due to noise uncertainty. Spectrum sharing will be an important technology in addressing lack of spectrum. Wireless users need to quickly sense and access idle bands to use shared bandwidth. The main unresolved problems of spectrum sensing are: (i) To identify minute gaps in the spectrum and to ensure that tight real-time digital signal processing (DSP) requirements are met, it must function even at very low latency. (ii) to find applications, its algorithms must be more precise and adaptable with various wireless bands and protocols. As far as we aware, there are no spectrum sensing techniques in the literature that could achieve both goals.

In this project, we provide DetectNet, a software framework or model designed for current wideband spectrum sensing. It employs real-time deep learning and integrates it with the baseband processing logic of the transceiver to recognize and display unused spectrum bands. A Long Short-Term Memory (LSTM) and a convolutional neural network is integrated known as DetectNet, which will automatically extract the information with the smaller number of I/Q samples. DetectNet, a deep learning model that works as a detection system is proposed, which will provide more accurate results than existing conventional spectrum sensing methods.

Keywords: deep learning, signal-to-noise ratio, long short-term memory, signal detection, licensed spectrum

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title of the base paper:

Deep learning for spectrum sensing

Journal name:

IEEE Wireless Communications Letters

Publisher:

IEEE Wireless Communications Letters

Year of Publishing:

2019

Indexed in:

IEEE Explore

Novelty of the Base paper

This paper proposes a novel scheme for spectrum sensing among primary and secondary users using cognitive radio. The main novelty of the proposed scheme lies in its ability to effectively classify the spectrum into occupied or unoccupied using the hypothesis:

$$\boldsymbol{y}(n) = \begin{cases} \boldsymbol{w}(n): & \mathscr{H}_0 \\ \boldsymbol{h}\boldsymbol{s}(n) + \boldsymbol{w}(n): & \mathscr{H}_1, \end{cases}$$

Specifically, the proposed scheme utilizes a combination of CNN and LSTM models to classify the spectrum into occupied and unoccupied.

Introduction

An SNR wall is an specific level of Signal-to-Noise (SNR) at which below this level conventional energy detector fails to functioning because of noise uncertainty.

In this, the authors proposed three distinct strategies to overcome the SNR wall: making use of primary user signals and lowering uncertainty of noise.

In this project, we first present a detector based on Deep Learning which is combination of convolutional long short-term deep neural networks (CLDNN), is useful for a variety of primary signal types and motivated by promising results.

It is important to note that the suggested detector doesn't require any further knowledge of the primary user signal or any noise density.

Simulation results proves that the suggested DL-based detection approaches perform noticeably better than the traditional methods.

Problem Formulation

A binary hypothesis testing problem for the detection of signal at the secondary user is formed depending on whether the original user is idle or busy.

$$y(n) = \begin{cases} w(n): & \mathscr{H}_0 \\ hs(n) + w(n): & \mathscr{H}_1, \end{cases}$$

Where h represents channel gain, and we are assuming it to be a constant throughout the period of sensing, y(n) represents the n-th received sample, s(n) represents primary user signal, w(n) represents added noise which follows the zero means circularly symmetric complex Gaussian (CSCG) distribution, and his channel gain.

the received signal's energy, normalized sample size of N and noise variance of 2w2Pd = 1Pmd is the test statistic for the conventional energy detector. Even at extremely low SNRs, a decent detector should achieve low Pf and Pmd. The SNR-wall and noise-only samples can estimate the noise density under particular performance requirements.

DL Based Detector

In this project, we first present a Deep Learning detector employing a Convolutional Long Short-Term Deep Neural Networks (CLDNN). It is used for a wide variety of primary signal types and it is motivated by promising results.

When it is used online, it is important to note that the suggested detector doesn't require any further knowledge about both primary signal and noise density.

The simulation findings demonstrate that the recommended DL-based detection approaches outperform the conventional techniques significantly.

Network Architecture Design

Since CLDNN works optimally in jobs requiring modulation recognition, we choose this style of architecture for our project.

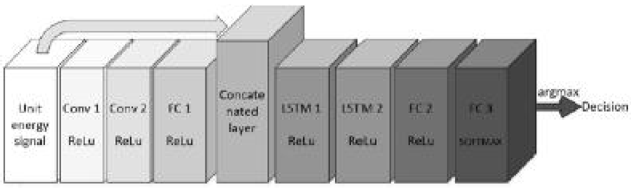

Figure 1: Network Architecture of DetectNet

Numerical simulations will also show how CLDNN is superior than other well-liked neural network tpologies.

It is obvious that a neural network model provides the best performance with 2 LSTM layers, 2 convolutional (Conv) layers, 1 fully connected (FC) layer after the Conv layers, and 2 FC layers after the LSTM layers.

All other layers use ReLu, but FC3 utilizes SOFTMAX. After each layer, dropout is employed to avoid overfitting.

Dataset generation and pre-processing

By RadioML2016.10a, a baseline dataset that is frequently used in modulation recognition tasks and digitally modulated signals of 8 different types at various positive SNRs, while the negative samples are following CSCG noises.

We are using standard split of ratio 3:1:1, the entire dataset is divided into 3 separate setsand it is for training, validation, and testing. Before training or inferring, energy normalization is carried out rather than just employing the received complex signal which is in time-domain.

TABLE I
HYPERPARAMETERS OF THE PROPOSED CLDNN

Hyperparameter	Value
Filters per Conv layer	60
Filter size	10
Cells per LSTM layer	128
Neurons per FC layer	128 & Sample length & 2
Optimizer	Adam
Initial learning rate	0.0003
Batch size	200
Dropout ratio	0.2

Three factors drive this: According to simulation results, (1) the influence of energy is found to be minimal, (2) the signals modulation structure can be better exposed without disturbance or any effect from signal energy, (3) even if the background noise changes, the energy independent model will produce general capability when it is working and (4) an energy independent model can be used to simulate the effects of energy.

TABLE II
DATASET PARAMETERS

Modulation scheme	BPSK,QPSK,8PSK,CPFSK QAM16,QAM64,GFSK,PAM4
Samples per symbol	8
Sample length	64, 128, 256, 512, 1024
SNR range	-20∼20dB in 1-dB increments
Training samples	48000
Validation samples	16000
Testing samples	16000

Customized two-stage training

Pf and Pd are two crucial performance metrics for signal identification that cannot be retrieved directly from the DL library.

For training the model to convergence, early stopping with six epochs are used.

The accuracy and validation loss both remain consistent according to the metrics trade-off feature, however, Pf and Pd at various SNRs change over time.

We first fix a Pf stop interval, proceed from the 1st stage's best model, and we stop training when Pf reaches it.

Applying the 2 stage training technique, we may somewhat regulate performance of detection by modifying the pre-set stop interval. One disadvantage of Deep Learning systems is exact performance control is absent.

The interval size option allows for a trade-off between training time and control precision.

A short interval results in more accurate performance control and lengthier training periods.

Simulation Results

The effectiveness of the suggested paradigm is illustrated by extensive simulation results. the effects of important factors like sample length and modulation scheme are examined.

1) Contrast With Other Networks: On QAM16 signals with 128 samples, Fig. No. 2 compare the proposed model's detection performance with those of several other well-known neural network models.

2) The effect of the modulation scheme is shown in Figure 3, which shows how well DetectNet detects throughout a range of modulation schemes with 128 samples.

3) Generalisation Capability: Fig. 4 shows how the suggested DetectNet can be generalized. We specifically examine how well a well-trained proposed network detects signals using various modulation schemes that are distinct from the training signals.

Comparing to energy detector, DetectNet will consistently gives 5dB improvements regardless of sample length.

System Design

For the derivation of probability of 2 hypotheses about primary signal, DetectNet was used locally.

It is fed into the fusion facility to undergo additional processing.

In this paper, a neural network with 3 FC layers is proposed to directly choose the appropriate fusion rule through training, in contrast to conventional sensing systems that integrate the hard decision data from faraway nodes using a pre-set rule.

The number of neurons in each FC layer is determined by thorough cross-validation and is 32, 8, and 2, respectively.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

MERITS

- Complexity is less
- Link reliability is improved
- Better utilization of spectrum is offered
- It is more efficient
- Lost is lowered
- Advanced network topologies are used
- Network architecture is simple
- Configuration and upgradation is easy

DEMERITS

- Automation is not complete and any changes to be implemented requires user intervention
- Multi band antenna is required
- It has a security concern: in cognitive radio, there are more chances for being hacked compared to conventional traditional methods. And hackers may get access to all information.
- Quality of Service is affected in cognitive radio
- Translation of observations into actions is a big challenge in this cognitive radio.

CHAPTER 3

SOURCE CODE

```
import tensorflow as tf

from tensorflow.keras.layers import Reshape, Conv2D, MaxPool2D, ZeroPadding2D, Dense,
Dropout, Activation, Flatten, GaussianNoise


import matplotlib.pyplot as plt

import numpy as np

import pandas as pd


import pickle


def digitizer(labels):


    unique_labels = np.unique(labels)

    label_dict = {}

    num = 1

    for i in unique_labels:

        label_dict[i] = num

        num += 1


    digit_label = []

    for i in labels:

        digit_label.append(label_dict[i])


    return label_dict,  digit_label




def onehot_encoder(L_dict,  labels):

    num_classes = len(L_dict)

    vector = np.zeros(num_classes)

    vector[L_dict[labels]-1] = 1
```

```
    return vector



def confusion_matrix_create (y_true, y_pred, labels_dict, title):

    labels = []
    for i in labels_dict.items():
        labels.append(i[0])
    y_true = np.argmax(y_true, axis =1)
    y_true = np.array(y_true) + 1
    y_pred = np.array(y_pred) + 1



    updated_pred = []
    updated_true = []

    for i in range(len(y_true)):

        for key,value in labels_dict.items():
            if value == y_true[i]:
                updated_true.append(key)

            if value == y_pred[i]:
                updated_pred.append(key)

    cm = confusion_matrix(updated_true,updated_pred, labels)
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    fig, ax = plt.subplots()
    fig.set_figheight(10)
    fig.set_figwidth(10)
```

```
    plt.xticks(ticks=[-1,0,1,2,3,4,5,6,7,8,9,10], rotation=45)
    plt.yticks(ticks=[-1,0,1,2,3,4,5,6,7,8,9,10], rotation=45)
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)
    ax.set_xticklabels([''] + labels)
    ax.set_yticklabels([''] + labels)
    ax.set (title=title,
            ylabel='True label',
            xlabel='Predicted label')
    fmt = '.2f'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    plt.show()

import numpy as np
import pickle

with open('/kaggle/input/radioml2016-deepsigcom/RML2016.10a_dict.pkl', "rb") as p:
    d = pickle.load(p, encoding='latin1')

classes = []
for i in d.keys():
    if i[0] not in classes:
        classes.append(i[0])

# creating class dictionary for strings to digits transformation.
label_dict,  digit_label = digitizer(classes)
```

```
SNRs = {}
for key in d.keys():
    if key not in SNRs:
        SNRs[key[1]] = []
SNRs.keys()



j = 0
for keys in d.keys():
    for arrays in d[keys]:
        # convert labels to one-hot encoders.
        SNRs[keys[1]].append([onehot_encoder(label_dict, keys[0]),np.array(arrays)])

outfile = open('dataset','wb')
pickle.dump(SNRs,outfile)
outfile.close()



outfile = open('class_dict','wb')
pickle.dump(label_dict,outfile)
outfile.close()

import pickle
import itertools
from random import shuffle

with open('dataset', 'rb') as file:
    data = pickle.load(file, encoding='Latin')
```

```
for key in data.keys():
    shuffle(data[key])

new_data = {'combined': []}
SNR_test = {}

for key in data.keys():
    train_len = int(0.9 * len(data[key]))
    new_data['combined'].append(data[key][:train_len])
    SNR_test[key] = data[key][train_len:]

new_data['combined'] = list(itertools.chain.from_iterable(new_data['combined']))

outfile = open('new_model_SNR_test_samples', 'wb')
pickle.dump(SNR_test, outfile)
outfile.close()

outfile = open('combined_SNR_data', 'wb')
pickle.dump(new_data, outfile)
outfile.close()

from keras.datasets import cifar10
from keras.utils import np_utils
from keras import metrics
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, LSTM, BatchNormalization
from keras import metrics
from keras.losses import categorical_crossentropy
from keras.optimizers import SGD
import pickle
import matplotlib.pyplot as plt
import numpy as np
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras import layers
from keras.callbacks import EarlyStopping


def CLDNN():
    model = Sequential()
    model.add(Conv2D(256, (1, 3), activation='relu', padding='same',
kernel_initializer='glorot_uniform', input_shape=(2, 128, 1)))
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid',  data_format=None))
    model.add(layers.Dropout(0.3))


    model.add(Conv2D(256, (2, 3), activation='relu', padding='same',
kernel_initializer='glorot_uniform'))
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid',  data_format=None))
    model.add(layers.Dropout(0.3))


    model.add(Conv2D(80, (1, 3), activation='relu', padding='same',
kernel_initializer='glorot_uniform'))
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid',  data_format=None))
    model.add(layers.Dropout(0.3))


    model.add(Conv2D(80, (1, 3), activation='relu', padding='same',
kernel_initializer='glorot_uniform'))
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid',  data_format=None))
    model.add(layers.Dropout(0.3))


    model.add(Reshape((2, 640)))


    model.add(LSTM(50, activation='relu'))
    model.add(layers.Dropout(0.3))


    model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
    model.add(layers.Dropout(0.3))
```

```
    model.add(Dense(11, activation='softmax', kernel_initializer='he_normal'))

    return model

def Robust_CNN():

    model = Sequential()
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same', input_shape=(2,128,1)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid',  data_format=None))
    model.add(layers.Dropout(.3))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid', data_format=None))
    model.add(layers.Dropout(.3))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid', data_format=None))
    model.add(layers.Dropout(.3))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(1, 2), padding='valid', data_format=None))
    model.add(layers.Dropout(.3))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(11, activation='softmax'))

    return model

def DNN():
    model = Sequential()
```

```
    model.add(Dense(512,activation='relu'))
    model.add(Dense(256,activation='relu'))
    model.add(Dense(128,activation='relu'))
    model.add(Dense(64,activation='relu'))
    model.add(Flatten())
    model.add(Dense(128,activation='relu'))
    model.add(Dense(11,activation='softmax'))
    return model

def Robust_LSTM():
    model = Sequential()
    model.add(LSTM(units=300,activation='relu',input_shape=(2,128)))
    model.add(BatchNormalization())
    model.add(layers.Dropout(.3))
    model.add(Reshape((2,150)))
    model.add(LSTM(units=200,activation='relu'))
    model.add(BatchNormalization())
    model.add(layers.Dropout(.3))
    model.add(Reshape((2,25)))
    model.add(layers.Dropout(.3))
    model.add(Dense(128,activation='relu'))
    model.add(Dense(11,activation='softmax'))
    return model

from sklearn.metrics import confusion_matrix
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.optimizers import Adam
import os
import pickle
import numpy as np
# Use this code only if you want to generate 20 different models corresponding to 20 SNR values
accuracies_All = []
```

```
confusion_matrices_All = []

for key in SNRs.keys():
    dataset = []
    labels = []

    for values in SNRs[key]:
        labels.append(values[0])
        dataset.append(values[1])

    print('Starting training for SNR:', key)

    N = len(dataset)
    shuffled_indeces = np.random.permutation(range(N))
    new_dataset = np.array(dataset)[shuffled_indeces,:,:]
    new_labels = np.array(labels)[shuffled_indeces,:]

    num_train = int(0.8*N)
    x_train = new_dataset[:num_train,:,:]
    y_train = new_labels[:num_train,:]

    num_val = int(0.1*len(x_train))

    x_val = x_train[:num_val,:,:]
    x_val = x_val.reshape(x_val.shape[0],x_val.shape[1],x_val.shape[2], -1)
    y_val = y_train[:num_val,:]

    x_train = x_train[num_val:,:,:]
    x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2], -1)
    y_train = y_train[num_val:,:]

    x_test = new_dataset[num_train:,:,:]
```

```
    x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2], -1)
    y_test = new_labels[num_train:,:]

    models = CLDNN()
    opt = Adam(learning_rate=0.0003)
    models.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

    num_epochs = 300

    # Checkpoint for models
    ckpt_folder = "cldnn_models/"
    ckpt_file_path = 'cldnn_model_SNR_{}'.format(key)
    if not os.path.exists(ckpt_folder):
        os.mkdir(ckpt_folder)
    model_ckpt_callback =
ModelCheckpoint(filepath=ckpt_folder+ckpt_file_path,monitor='val_loss', mode='min',
save_best_only=True)
    reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=20,
verbose=1, epsilon=1e-4, mode='min')

    history = models.fit(x_train,
                    y_train,
                    epochs=num_epochs,
                    batch_size=200,
                    callbacks = [reduce_lr_loss, model_ckpt_callback],
                    validation_data=(x_val, y_val))
    loss, acc = models.evaluate(x_test, y_test, verbose=2)
    predicted_data = models.predict(x_test)
    accuracies_All.append([acc, key])
    print('accuracy =', acc)
    res = np.argmax(predicted_data, 1)
    y_test_res = np.argmax(y_test, 1)
    results = confusion_matrix((y_test_res+1), (res+1))
```

```
    confusion_matrices_All.append([results, key])

dic = dict()
for i in accuracies_All_lstm:
    dic[i[1]] = i[0]

for i in snr:
    acc.append(dic[i])
acc = acc[:20]

import matplotlib.pyplot as plt
plt.plot(snr,acc,color='green')
plt.xlabel("SNR")
plt.ylabel("ACCURACY")
plt.title("Accuracy vs Snr for CLDNN model")
plt.show()

snr = []
acc = []
for i in dic:
    snr.append(i)
snr.sort()

from sklearn.metrics import confusion_matrix
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.optimizers import Adam
import os
import pickle
import numpy as np
accuracies_All = []
confusion_matrices_All = []
```

```
for key in SNRs.keys():
    dataset = []
    labels = []

    for values in SNRs[key]:
        labels.append(values[0])
        dataset.append(values[1])

    print('Starting training for SNR:', key)

    N = len(dataset)
    shuffled_indeces = np.random.permutation(range(N))
    new_dataset = np.array(dataset)[shuffled_indeces,:,:]
    new_labels = np.array(labels)[shuffled_indeces,:]

    num_train = int(0.8*N)
    x_train = new_dataset[:num_train,:,:]
    y_train = new_labels[:num_train,:]

    num_val = int(0.1*len(x_train))

    x_val = x_train[:num_val,:,:]
    x_val = x_val.reshape(x_val.shape[0],x_val.shape[1],x_val.shape[2], -1)
    y_val = y_train[:num_val,:]

    x_train = x_train[num_val:,:,:]
    x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2], -1)
    y_train = y_train[num_val:,:]

    x_test = new_dataset[num_train:,:,:]
    x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2], -1)
    y_test = new_labels[num_train:,:]
```

```
models = DNN()
opt = Adam(learning_rate=0.0003)
models.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

num_epochs = 300

# Checkpoint for models
ckpt_folder = "cldnn_models/"
ckpt_file_path = 'cldnn_model_SNR_{}'.format(key)
if not os.path.exists(ckpt_folder):
    os.mkdir(ckpt_folder)
model_ckpt_callback =
ModelCheckpoint(filepath=ckpt_folder+ckpt_file_path,monitor='val_loss', mode='min',
save_best_only=True)
reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=20,
verbose=1, epsilon=1e-4, mode='min')

history = models.fit(x_train,
                y_train,
                epochs=num_epochs,
                batch_size=200,
                callbacks = [reduce_lr_loss, model_ckpt_callback],
                validation_data=(x_val, y_val))
loss, acc = models.evaluate(x_test, y_test, verbose=2)
predicted_data = models.predict(x_test)
accuracies_All.append([acc, key])
print('accuracy =', acc)
res = np.argmax(predicted_data, 1)
y_test_res = np.argmax(y_test, 1)
results = confusion_matrix((y_test_res+1), (res+1))
confusion_matrices_All.append([results, key])
```

```
from sklearn.metrics import confusion_matrix
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.optimizers import Adam
import os
import pickle
import numpy as np
# Use this code only if you want to generate 20 different models corresponding to 20 SNR values
accuracies_All = []
confusion_matrices_All = []


for key in SNRs.keys():
    dataset = []
    labels = []

    for values in SNRs[key]:
        labels.append(values[0])
        dataset.append(values[1])

    print('Starting training for SNR:', key)

    N = len(dataset)
    shuffled_indeces = np.random.permutation(range(N))
    new_dataset = np.array(dataset)[shuffled_indeces,:,:]
    new_labels = np.array(labels)[shuffled_indeces,:]

    num_train = int(0.8*N)
    x_train = new_dataset[:num_train,:,:]
    y_train = new_labels[:num_train,:]

    num_val = int(0.1*len(x_train))

    x_val = x_train[:num_val,:,:]
```

```
   x_val = x_val.reshape(x_val.shape[0],x_val.shape[1],x_val.shape[2], -1)
   y_val = y_train[:num_val,:]

   x_train = x_train[num_val:,:,:]
   x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2], -1)
   y_train = y_train[num_val:,:]

   x_test = new_dataset[num_train:,:,:]
   x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2], -1)
   y_test = new_labels[num_train:,:]

   models = Robust_CNN()
   opt = Adam(learning_rate=0.0003)
   models.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

   num_epochs = 300

   # Checkpoint for models
   ckpt_folder = "cldnn_models/"
   ckpt_file_path = 'cldnn_model_SNR_{}'.format(key)
   if not os.path.exists(ckpt_folder):
      os.mkdir(ckpt_folder)
   model_ckpt_callback = 
ModelCheckpoint(filepath=ckpt_folder+ckpt_file_path,monitor='val_loss', mode='min', 
save_best_only=True)
   reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=20, 
verbose=1, epsilon=1e-4, mode='min')

   history = models.fit(x_train,
                 y_train,
                 epochs=num_epochs,
                 batch_size=200,
                 callbacks = [reduce_lr_loss, model_ckpt_callback],
```

```
                    validation_data=(x_val, y_val))
    loss, acc = models.evaluate(x_test, y_test, verbose=2)
    predicted_data = models.predict(x_test)
    accuracies_All.append([acc, key])
    print('accuracy =', acc)
    res = np.argmax(predicted_data, 1)
    y_test_res = np.argmax(y_test, 1)
    results = confusion_matrix((y_test_res+1), (res+1))
    confusion_matrices_All.append([results, key])


from sklearn.metrics import confusion_matrix
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.optimizers import Adam
import os
import pickle
import numpy as np
# Use this code only if you want to generate 20 different models corresponding to 20 SNR
values
accuracies_All = []
confusion_matrices_All = []


for key in SNRs.keys():
    dataset = []
    labels = []


    for values in SNRs[key]:
        labels.append(values[0])
        dataset.append(values[1])


    print('Starting training for SNR:', key)


    N = len(dataset)
    shuffled_indeces = np.random.permutation(range(N))
```

```
new_dataset = np.array(dataset)[shuffled_indeces,:,:]
new_labels = np.array(labels)[shuffled_indeces,:]

num_train = int(0.8*N)
x_train = new_dataset[:num_train,:,:]
y_train = new_labels[:num_train,:]

num_val = int(0.1*len(x_train))

x_val = x_train[:num_val,:,:]
x_val = x_val.reshape(x_val.shape[0],x_val.shape[1],x_val.shape[2], -1)
y_val = y_train[:num_val,:]

x_train = x_train[num_val:,:,:]
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2], -1)
y_train = y_train[num_val:,:]

x_test = new_dataset[num_train:,:,:]
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2], -1)
y_test = new_labels[num_train:,:]

models = Robust_LSTM()
opt = Adam(learning_rate=0.0003)
models.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

num_epochs = 300

# Checkpoint for models
ckpt_folder = "cldnn_models/"
ckpt_file_path = 'cldnn_model_SNR_{}'.format(key)
if not os.path.exists(ckpt_folder):
    os.mkdir(ckpt_folder)
```

```
    model_ckpt_callback =
ModelCheckpoint(filepath=ckpt_folder+ckpt_file_path,monitor='val_loss', mode='min',
save_best_only=True)
    reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=20,
verbose=1, epsilon=1e-4, mode='min')


    history = models.fit(x_train,
                         y_train,
                         epochs=num_epochs,
                         batch_size=200,
                         callbacks = [reduce_lr_loss, model_ckpt_callback],
                         validation_data=(x_val, y_val))
    loss, acc = models.evaluate(x_test, y_test, verbose=2)
    predicted_data = models.predict(x_test)
    accuracies_All.append([acc, key])
    print('accuracy =', acc)
    res = np.argmax(predicted_data, 1)
    y_test_res = np.argmax(y_test, 1)
    results = confusion_matrix((y_test_res+1), (res+1))
    confusion_matrices_All.append([results, key])


dic = dict()
for i in accuracies_All_lstm:
    dic[i[1]] = i[0]


for i in snr:
    acc.append(dic[i])
acc = acc[:20]


import matplotlib.pyplot as plt
plt.plot(snr,acc,color='green')
plt.xlabel("SNR")
plt.ylabel("ACCURACY")
```

```
plt.title("Accuracy vs Snr for CLDNN model")
plt.show()

snr = []
acc = []
for i in dic:
   snr.append(i)
snr.sort()



dic = dict()
for i in accuracies_All_lstm:
   dic[i[1]] = i[0]

for i in snr:
   acc.append(dic[i])
acc = acc[:20]

import matplotlib.pyplot as plt
plt.plot(snr,acc,color='green')
plt.xlabel("SNR")
plt.ylabel("ACCURACY")
plt.title("Accuracy vs Snr for CLDNN model")
plt.show()

snr = []
acc = []
for i in dic:
   snr.append(i)
snr.sort()
```

```
dic = dict()
for i in accuracies_All_lstm:
    dic[i[1]] = i[0]

for i in snr:
    acc.append(dic[i])
acc = acc[:20]

import matplotlib.pyplot as plt
plt.plot(snr,acc,color='green')
plt.xlabel("SNR")
plt.ylabel("ACCURACY")
plt.title("Accuracy vs Snr for CLDNN model")
plt.show()
```

CHAPTER 4

SNAPSHOTS

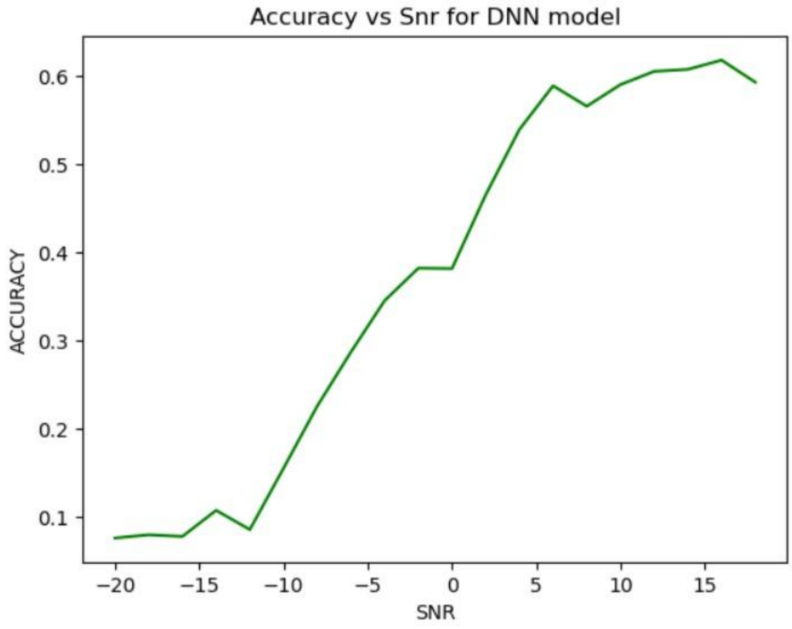

Figure 2: DNN Model

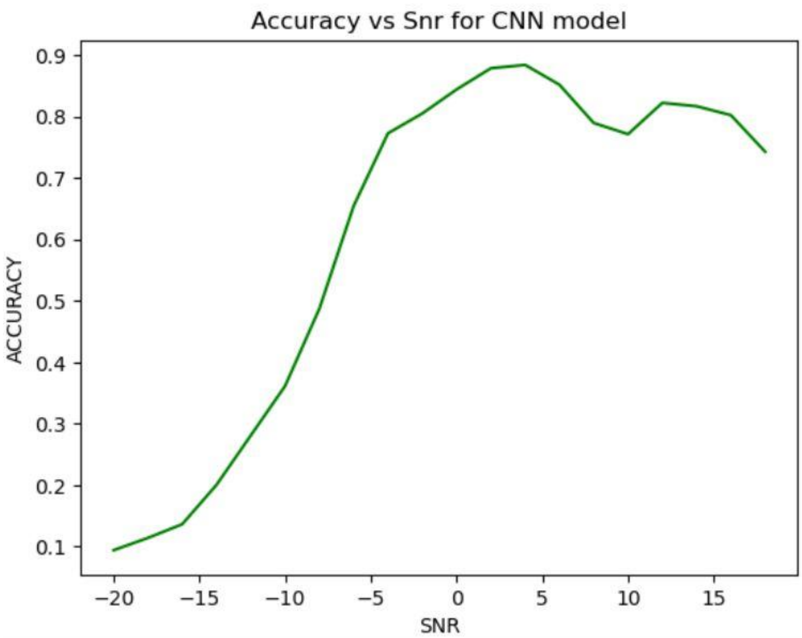

Figure 3: CNN Model

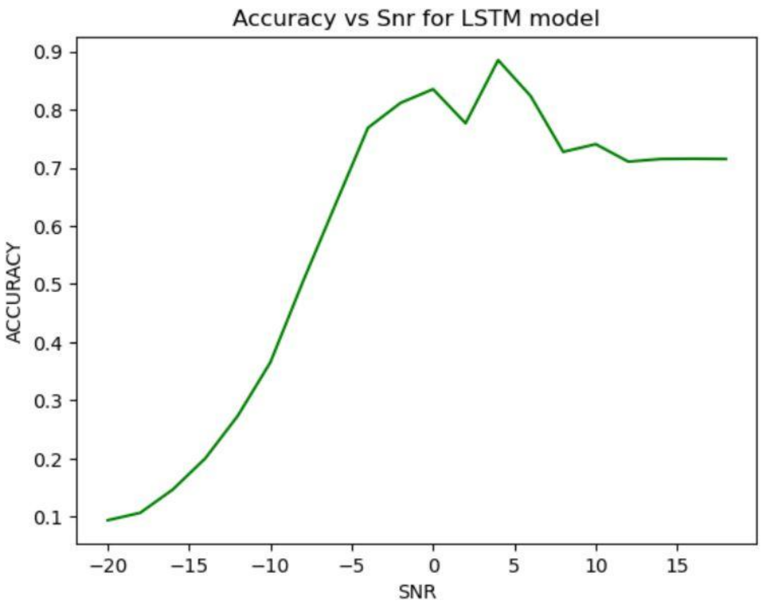

Figure 4: LSTM Model

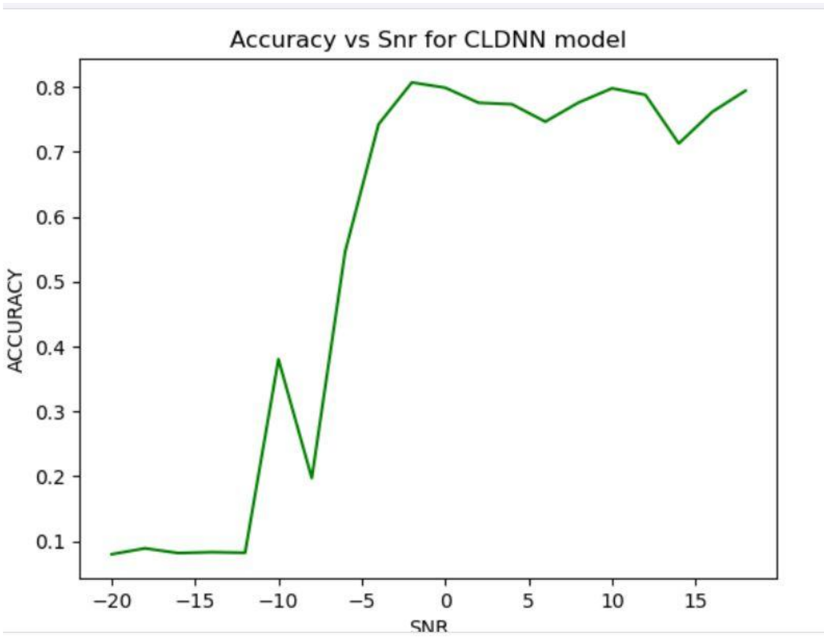

Figure 5: CLDNN Model

CHAPTER 5

CONCLUSION AND FUTURE PLANS

We have put out a brand-new DL-based signal detector called DetectNet that takes advantage of the built-in structural information included in modulated signals.

It is demonstrated that a notable performance improvement over the standard energy detector was practical.

The Deep Learning-based detector exhibits high generalization properties to related modulation schemes and is insensitive to changes in modulation order.

A cooperative detection method called SoftCombinationNet that is based on deep learning is described in order to use the soft data from dispersed sensing nodes. It has been shown to simultaneously achieve low Pf and high Pd.

It cannot use the decision-making confidence information of each node.

Different nodes' priorities are not used.

We provide a cooperative detection method based on DL that implicitly makes use of this soft information.

The proposed CLDNN model doesn't produce the expected result for low SNR values. In the future a new model can be built to overcome SNR wall and to classify low SNR valued signals with high accuracy

CHAPTER 6

REFERENCES

1. S. Zheng, S. Chen, P. Qi, H. Zhou and X. Yang, "Spectrum sensing based on deep learning classification for cognitive radios," in *China Communications*, vol. 17, no. 2, pp. 138-148, Feb. 2020, doi: 10.23919/JCC.2020.02.012.
2. M. Ben mohammed mahieddine, N. Mellah, A. Bassou, M. Khelifi and S. A. Chouakri, "Implementation of CNN-Inception Deep Learning for Cognitive Radio Based on Modulation Classifications," *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, Meknes, Morocco, 2022, pp. 1-5, doi: 10.1109/IRASET52964.2022.9738405.
3. A. Sundriyal and A. Baghel, "“CNN based Cognitive Spectrum Sensing with Optimization”," 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2020, pp. 1537-1540, doi: 10.1109/ICECA49313.2020.9297438.
4. S. Zhang, Z. Xu, L. Tian and X. Yang, "A Spectrum Sensing Method Based on CNN-LSTM Deep Neural Network," 2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC), Beijing, China, 2021, pp. 186-190, doi: 10.1109/IC-NIDC54101.2021.9660470.
5. T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, “Deep learning for wireless physical layer: Opportunities and challenges,” *China Commun.*, vol. 14, no. 11, pp. 92–111, Nov. 2017.
6. Y.-C. Liang, Y. Zeng, E. C. Y. Peh, and A. T. Hoang, “Sensing throughput tradeoff for cognitive radio networks,” *IEEE Trans. Wireless Commun.*, vol. 7, no. 4, pp. 1326–1337, Apr. 2008.
7. F. Azmat, Y. Chen, and N. Stocks. “Analysis of spectrum occupancy using machine learning algorithms,” *IEEE Trans. Vehi. Tech.*, vol. 65, no. 9, pp. 6853–6860, Sep. 2016.
8. S. Atapattu, C. Tellambura, and H. Jiang, “Conventional energy detector,” in *Energy Detection for Spectrum Sensing in Cognitive Radio* (Spring Briefs in Computer Science). New York, NY, USA: Springer, 2014, pp. 11–26.
9. A. Mariani, A. Giorgetti, and M. Chiani, “Effects of noise power estimation on energy detection for cognitive radio applications,” *IEEE Trans. Commun.*, vol. 59, no. 12, pp. 3410–3420, Dec. 2011.
10. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
11. R. Tandra and A. Sahai, “SNR walls for signal detection,” *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 1, pp. 4–17, Feb. 2018.

CHAPTER 7

APPENDIX - BASE PAPER

Title:

Deep Learning for Spectrum Sensing

Citation:

TY - JOUR

TI - Deep Learning for Spectrum Sensing

T2 - IEEE Wireless Communications Letters

SP - 1727

EP - 1730

AU - J. Gao, X. Yi, C. Zhong, X. Chen, Z. Zhang

PY - 2019

DO - 10.1109/LWC.2019.2939314

JO - IEEE Wireless Communications Letters

https://ieeexplore.ieee.org/document/8824091