



Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Radijo siųstuvų klasifikavimo, naudojant Hilberto-Huango transformacijos metodą ir neuroninius tinklus, tyrimas

Baigiamasis magistro projektas

Aretas Paulauskas

Projekto autorius

Prof. Darius Gailius

Vadovas

Kaunas, 2024



Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Radio siųstuvų klasifikavimo, naudojant Hilberto-Huango transformacijos metodą ir neuroninius tinklus, tyrimas

Baigiamasis magistro projektas

Elektronikos inžinerija (6211EX012)

Aretas Paulauskas

Projekto autorius

Prof. Darius Gailius

Vadovas

Doc. Darius Kybartas

Recenzentas

Kaunas, 2024



Kauno technologijos universitetas

Elektros ir elektronikos

Aretas Paulauskas

Radijo siūstuvų klasifikavimo, naudojant Hilberto-Huango transformacijos metodą ir neuroninius tinklus, tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdamas kitų asmenų autoriaus ar kitų teisių, laikydamasis Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;
2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąrašė;
3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs;
4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalintas iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Aretas Paulauskas

Patvirtinta elektroniniu būdu

Paulauskas Aretas. Radijo siųstuvų klasifikavimo, naudojant Hilberto-Huango transformacijos metodą ir neuroninius tinklus, tyrimas. Magistro baigiamasis projektas / vadovas prof. Darius Gailius; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, inžinerijos mokslai.

Reikšminiai žodžiai: Hilberto-Huango transformacijos spektras, pereinamieji signalai, skaitmeninių signalų apdorojimas, gilieji neuroniniai tinklai.

Kaunas, 2024. 99 p.

Santrauka

Šio darbo tikslas yra išanalizuoti radijo siųstuvų siunčiamus signalus ir pagal šių signalų pradžios atkarpas atpažinti ir klasifikuoti specifinį radijo siųstuvą.

Pirmoje darbo dalyje apžvelgiami įvairūs moksliniai darbai, juose naudoti skaitiniai metodai ir algoritmai. Pateikiamas radijo siųstuvų klasifikavimo palyginimas tarp mašininio mokymosi ir neuroninių tinklų metodų – analizuojamas skirtingų ištrauktų požymių poveikis klasifikavimo tikslumui. Taip pat, analizuojami ir aprašomi literatūroje naudojami radijo signalo pradžios aptikimo algoritmai ir kompleksinio signalo analizės būdai.

Antroji dalis susideda iš duomenų rinkimo architektūros ir įrangos aprašymo – pateikiamos diagramos bei detalizuojama naudota įranga.

Trečioje dalyje analizuojami gauti duomenys – analizuojamos radijo siųstuvų FSK, GFSK ir LoRa moduliacijos naudojant Hilberto–Huango transformacijos metodą, kuris palyginamas su trumpalaikė Furjė transformacija. Aprašomas ir atliekamas papildomas surinktų duomenų išplėtimas – gaunamas mokymosi duomenų rinkinys.

Ketvirtoji dalis pateikia neuroninių tinklų klasifikavimo eigą ir rezultatus. Pateikiami trijų tinklų palyginimai: ResNet-34, VGG-16 ir literatūros autorių pasiūlytas, ResNet pagrindu sukurtas, CNN tinklas.

Paulauskas Aretas. Research of Radio Transmitter Classification Using Hilbert-Huang Transform Method and Neural Networks. Master's Final Degree Project / supervisor Prof. Darius Gailius; Faculty of Electrical and Electronic Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics Engineering, Engineering Sciences.

Keywords: Hilbert - Huang transformation spectrum, transient signals, digital signal processing, machine learning, deep learning.

Kaunas, 2024. 99 p.

Summary

The main topic is to devise an algorithm to classify various radio transmitters based on their transient signals.

In the first part, a review is made on a diverse set of other researchers work, used algorithms and calculations. A comparison is made between machine and deep learning – a short investigation is also done on how various extracted data features impact the classification performance of trained networks. Also, a transient signal detection algorithms are reviewed as well as various ways to analyze a complex signal.

The second part consists of data collection system as well as equipment used in the process. Simplified diagrams are given and the used equipment is described.

The third part consists of the captured data analysis – various modulation schemes were analysed, including FSK, GFSK and LoRa. The analysis was made using Hilbert-Huang transform method and compared to Short – Time Fourier Transform. Data augmentation technique is used to increase the training data and make the trained network more robust.

The fourth part is where neural network classification takes place. A comparison between three different networks is made: ResNet-34, VGG-16 and a CNN network taken from analyzed authors, which was created on ResNet basis.

Turinys

Paveikslų sąrašas	8
Santrumpų ir terminų sąrašas	10
Įvadas.....	11
1. Literatūros apžvalga	12
1.1. Radijo siųstuvų klasifikavimo metodų apžvalga	14
1.1.1. Mašininio mokymosi algoritmų panaudojimas	15
1.1.2. Giliojo mokymosi algoritmų panaudojimas	18
1.2. Giliųjų neuroninių tinklų radijo siųstuvų klasifikavimo požymiai.....	21
1.3. Pereinamojo signalo pradžios nustatymas.....	22
1.4. Pereinamojo signalo kompleksinė analizė.....	24
1.5. Literatūros analizės apibendrinimas	25
2. Duomenų rinkimo architektūra ir įranga	26
2.1. Darbe naudojama įranga.....	26
2.1.1. Duomenų rinkimui naudojama įranga	26
2.1.2. Duomenų apdorojimui naudojama įranga	27
2.2. Duomenų rinkimo sistemos aprašymas	28
2.3. Eksperimente naudoti radijo siųstuvai.....	30
3. Surinktų radijo siųstuvų duomenų analizė	31
3.1. Analizuojamų duomenų apžvalga	31
3.2. Pereinamųjų signalų analizė	33
3.3. Pereinamųjų signalų laiko – dažnio – energijos pasiskirstymo analizė	41
3.4. Pereinamųjų signalų duomenų išplėtimas	52
3.5. Duomenų analizės apibendrinimas.....	56
4. Duomenų klasifikavimo tyrimas ir metodai.....	57
4.1. Konvoliucinis neuroninis tinklas	57
4.2. Radijo siųstuvų klasifikavimas	59
4.2.1. „Flatten Free“ CNN tinklas	60
4.2.2. VGG-16 tinklas	61
4.2.3. ResNet-34 tinklas	62
4.2.4. Paprastas CNN tinklas	64
4.3. Moduliacijos klasifikavimo tyrimas	65
4.4. Klasifikavimo rezultatų apibendrinimas.....	66
Išvados	68
Literatūros sąrašas	69
Priedai.....	72
1 priedas. Surinktų siųstuvų duomenų spektrogramos	72
2 priedas. Pagalbinės funkcijos	74
3 priedas. Duomenų nuskaitymas.....	80
4 priedas. Pereinamojo signalo išgavimas iš pilno signalo	82
5 priedas. Hilberto-Huango transformacijos spektro skaičiavimas.....	86
6 priedas. Gilaus mokymosi pagalbinės funkcijos	90
7 priedas. Gilaus mokymosi modelių sudarymo kodas	93
8 priedas. Gilaus mokymosi struktūrinis kodas.....	98

Lentelių sąrašas

1 lentelė. Pereinamosios ir stabilios signalo dalies panaudojimo klasifikavime, tikslumo įvertinimas [13]	15
2 lentelė. Bluetooth signalo duomenų analizei panaudoti statistiniai metodai [17]	16
3 lentelė. Mašininio mokymosi algoritmų tikslumo palyginimas esant skirtingam valdiklių kiekiui [17]	17
4 lentelė. Modelių tikslumas naudojant duomenų praplėtimą esant skirtingoms SNR reikšmėms klasifikuojant 10 skirtingų LoRa įrenginių naudojant SF7, SF8 ir SF9 LoRa konfigūracijas [20]... 20	
5 lentelė. Pereinamojo signalo pradžios metodų palyginimas [22][23]	22
6 lentelė. Pereinamojo signalo pradžios metodų privalumai ir trūkumai [22][23].....	22
7 lentelė. JCG401 antenos parametrai	26
8 lentelė. Pluto SDR imtuvo nustatyti pagrindiniai parametrai duomenų rinkimui	28
9 lentelė. Darbe naudotų radijo siųstuvų techninės charakteristikos.....	30
10 lentelė. RFM22B ir RFM69HW eksperimente naudotos moduliacijos ir jų parametrai.....	31
11 lentelė. RFM95HW eksperimente naudotos moduliacijos ir jų parametrai	31
12 lentelė. RFM22B ir RFM69HW pereinamųjų signalų vidutinės trukmės.....	40
13 lentelė. RFM95HW pereinamųjų signalų vidutinės trukmės	41
14 lentelė. Originalių duomenų apskaičiuotas vidutinis SNR, imtis – 750 paketų	53
15 lentelė. Radijo siųstuvų išplėstų ir originalių duomenų kiekių apibendrinimas.....	56
16 lentelė. „Flatten Free“ CNN tinklo modelio struktūra [20]	60
17 lentelė. VGG-16 modelio struktūra [33].....	61
18 lentelė. Paprastas trijų sluoksnių CNN tinklas	64
19 lentelė. RS klasifikavimui naudotų neuroninių tinklų modelių apibendrinti rezultatai.....	66
20 lentelė. Radijo siųstuvų įrašytų paketų spektrogramos	72

Paveikslų sąrašas

1 pav. Supaprastinta heterodino siųstuvo blokinė diagrama	12
2 pav. Įtampa valdomo dažnio generatoriaus blokinė diagrama	12
3 pav. RFM22/23B modulio schema [10]	13
4 pav. WiFi normalizuotas paketas su pažymėtais segmentais: triukšmas, signalo pereinamoji ir stabili dalis [12].....	14
5 pav. Autorių analizuojamų požymių vertė klasifikavimo procese naudojant mašininio mokymosi tinklą – SVM [17].....	16
6 pav. Darbe nagrinėtų skirtingų įrenginių centrinio dažnio poslinkio vertės [18].....	19
7 pav. Požymių ištraukimo proceso vizualizacija [21].....	20
8 pav. Įrenginio identifikavimas naudojant ištrauktus požymius [21]	21
9 pav. Akustinio signalo (plona linija) su jo energijos kreive (stora linija) minimumas [24].....	23
10 pav. Pluto SDR siųstuvas – imtuvas.....	26
11 pav. Duomenų rinkimo sistemos supaprastinta blokinė schema.	27
12 pav. Pilna duomenų rinkimo sistemos schema.....	28
13 pav. Vidutiniai PCA ištrauktų požymių klasifikavimo rezultatai esant skirtingiems diskretizavimo dažniams [27]	29
14 pav. „GNU Radio“ programinės įrangos blokų diagrama Pluto SDR valdymui (duomenų įrašymui).....	29
15 pav. Skirtingų radijo siųstuvų paketų pavyzdžiai laiko srityje.....	32
16 pav. Skirtingų radijo siųstuvų, atsitiktinai parinktų paketų, PSD spektras	33
17 pav. Pereinamojo signalo ištraukimo iš laiko srities duomenų algoritmas	34
18 pav. Pereinamojo signalo pabaigos nustatymas, kai įvestis yra disperijos reikšmių masyvas.....	36
19 pav. RFM22B radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija	37
20 pav. RFM69HW radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija	37
21 pav. RFM95HW radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija	37
22 pav. RFM22B radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai.....	38
23 pav. RFM69HW radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai.....	39
24 pav. RFM95HW radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai.....	39
25 pav. RFM22B, RFM69HW ir RFM95HW radijo siųstuvų pereinamieji signalai viename grafike	40
26 pav. STFT raiškos palyginimas. Kairėje – geresnė laiko raiška, dešinėje – geresnė dažninė raiška	41
28 pav. RFM22B radijo siųstuvo duomenų paketo laiko srities signalų.....	42
27 pav. RFM22 siųstuvo Hilberto-Huango ir STFT transformacijų palyginimas.....	42
29 pav. RFM69HW siųstuvo Hilberto–Huango ir STFT transformacijų palyginimas	43
31 pav. RFM69HW siųstuvo pereinamosios dalies momentinis dažnis esant FSK moduliacijai.....	44
30 pav. RFM22B siųstuvo pereinamosios dalies momentinis dažnis esant FSK moduliacijai.....	44
32 pav. RFM22B RS Hilberto-Huango FSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas.....	45
33 pav. RFM22B RS Hilberto-Huango GFSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas.....	45

34 pav. RFM69HW radijo siųstuvo FSK moduliacijos duomenų paketo laiko srities signalų komponentės – IMF.....	46
35 pav. RFM69HW RS Hilberto-Huango FSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas.....	47
36 pav. RFM69HW RS Hilberto-Huango GFSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas ...	47
37 pav. RFM95HW radijo siųstuvo duomenų paketo IMF, kai plotis 500 kHz.....	48
38 pav. RFM95HW radijo siųstuvo duomenų paketo IMF, kai plotis 125 kHz.....	48
39 pav. RFM95HW RS Hilberto-Huango LoRa moduliacijos spektras (kai plotis – 125 kHz), kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas.....	49
40 pav. RFM95HW RS Hilberto-Huango LoRa moduliacijos spektras (kai plotis – 31,25 kHz), kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas.....	50
42 pav. RFM69HW siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms.....	51
41 pav. RFM22 siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms.....	51
43 pav. RFM95HW siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms.....	52
44 pav. Laiko srityje pateikiamo pereinamojo signalo pavyzdys esant skirtingoms SNR reikšmėms.....	53
45 pav. RFM69HW siųstuvo, skirtingų SNR reikšmių, pereinamieji signalai.....	54
46 pav. RFM22 siųstuvo skirtingų SNR reikšmių, pereinamieji signalai.....	54
47 pav. RFM95HW siųstuvo, skirtingų SNR reikšmių, pereinamieji signalai.....	55
48 pav. Pavyzdinė neuroninio tinklo sandaros iliustracija [28].....	57
49 pav. Šašūkos operacijos pavyzdys [29].....	58
50 pav. <i>ReLU</i> aktyvacijos funkcija [30].....	58
51 pav. Šašūkos operacijos ir kraštų užpildymo pavyzdys [29].....	58
52 pav. Maksimalus telkimas (angl. <i>Max pooling</i>) [29].....	59
53 pav. Liekanos blokas [31].....	59
54 pav. „Flatten Free“ CNN tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b).....	61
55 pav. VGG-16 tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b)	62
57 pav. ResNet-34 tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b).....	63
56 pav. ResNet-34 modelio struktūra [34].....	63
58 pav. Paprasto CNN tinklo apmokymo rezultatai: klaidų matrica (a) ir praradimo funkcijos kitimas (b).....	64
59 pav. Moduliacijos klasifikavimo CNN tinklo praradimo funkcijos kitimas.....	65
60 pav. Moduliacijos klasifikacijos klaidų matrica.....	66

Santrumpų ir terminų sąrašas

Santrumpos:

RS - radijo siųstuvas;

Terminai:

AWGN – (angl. *Additive White Gaussian Noise*) sudėtinis baltasis Gauso triukšmas, turi normalųjį skirstinį, triukšmo vidurkis yra nulis;

BB – (angl. *Baseband*) dar nmoduluotas signalas;

CNN – (angl. *Convolutional Neural Network*) sąūkos neuroninis tinklas yra dirbtinio neuroninio tinklo rūšis, naudojantis sąūkos operaciją;

FFT – (angl. *Fast Fourier Transform*) algoritmas, skirtas laiko srities signalo transformacijai į dažnių sritį;

HHT – (angl. *Hilbert – Huang Transform*) Hilberto–Huango transformacija;

IoT – (angl. *Internet of things*) daiktų internetas;

LO – (angl. *Local oscillator*) vietinis dažnio generatorius, dažnai naudojamas su maišytuvu, kad užkeltų tarpinį dažnį į išsiunčiamą dažnį. Tikslumas matuojamas *ppm* vienetais;

LoRa – (angl. *Long range*) radijo moduliavimo sistema, kuri naudoja dažnio išplitimą;

RSSI – (angl. *Received Signal Strength Indicator*) gauto signalo stiprumas;

SDR – (angl. *Software defined radio*) radijo sistema, kurios parametrai gali būti konfigūruojami naudojant programinę įrangą;

STFT – (angl. *Short time Fourier Transform*) trumpalaikė Furjė transformacija;

TFED – (angl. *Time – frequency energy distribution*) laiko – dažnio – energijos pasiskirstymas;

Įvadas

Spartėjantis daiktų interneto įrenginių kiekis palaipsniui vis labiau teršia kompleksinį radijo dažnių spektrą. Radijo dažnių juosta yra padalinta į licencijuotas ir nelicencijuotas, limituoto pralaidumo dažnių juostas, kurios atskiriamos pagal skirtingus įrenginių panaudojimo atvejus – tampa ypač svarbu kuo efektyviau išnaudoti radijo spektro resursus [1].

Efektyvus spektro padalijimas yra ypatingos svarbos nelicencijuotuose ISM kanaluose, kuriuose stebimas didelis spektro užterštumas dėl besiplečiančio IoT įrenginių kiekio. Našus paskirstymas gali būti įgyvendinamas tik tuo atveju, jei dažnių spektras yra išanalizuotas, o spektre veikiantys įrenginiai ar bevielio ryšio protokolai teisingai suklasifikuoti. Radijo siųstuvų ar bevielio ryšio klasifikavimas gali būti plačiai panaudojamas ir karo pramonėje: signalų perėmimas, trukdymo (angl. *Jamming*) identifikavimas ar priešiško įrenginio aptikimas ir identifikavimas.

Radijo siųstuvų klasifikavimą galima išskirti į dvi pagrindines grupes: pereinamojo arba stabilaus signalo panaudojimą, kur stabilus signalas yra koduoti ir moduluoti duomenys su tam tikru, bevielio ryšio protokolo, apvalkalu.

Pirmieji klasifikavimo metodai buvo kuriami tradiciniais modeliais grįstais mašininio mokymosi sprendimais, naudojant ekspertų žinias, pavyzdžiui, Bajeso tinklus [2][3]. Išstobulėjus duomenų mokslui, atpažinimo metodai tapo didelių duomenų kiekio reikalaujančiais mašininio mokymosi metodais, tokiais kaip gilusis mašininis mokymasis [4][5]. Perėjimas nuo ekspertų nustatomų ir apskaičiuojamų požymių į vien tik dideliais duomenų kiekiais grįstą požymių išskaičiavimą naudojant neuroninius tinklus, leidžia lengviau klasifikuoti vis didėjančius radijo siųstuvų kiekius, o tai tampa ypač svarbu dėl naujai sukurtų daiktų interneto įrenginių su skirtingais protokolais ir sugebėjimais.

Sąsukų neuroniniai tinklai (angl. *Convolutional Neural Network, CNN*) dėl daugelio sluoksnių struktūros sugeba automatiškai atrinkti geriausius požymius bei gali veikti su daugiau nei vienos dimensijos duomenimis, pavyzdžiui, analizuojant signalo laikinę – dažninę charakteristiką [4][6]. Duomenimis grindžiamas automatinis aukšto lygio savybių išgavimas ženkliai sumažina aukšto lygio ekspertų žinių poreikį ir leidžia lengviau prisitaikyti prie naujų technologijų.

Šio darbo pagrindinis tikslas – giluminio neuroninio tinklo sukūrimas, kuris galėtų būti naudojamas skirtingų radijo siųstuvų klasifikavimui pagal jų skleidžiamus radijo dažnių signalus.

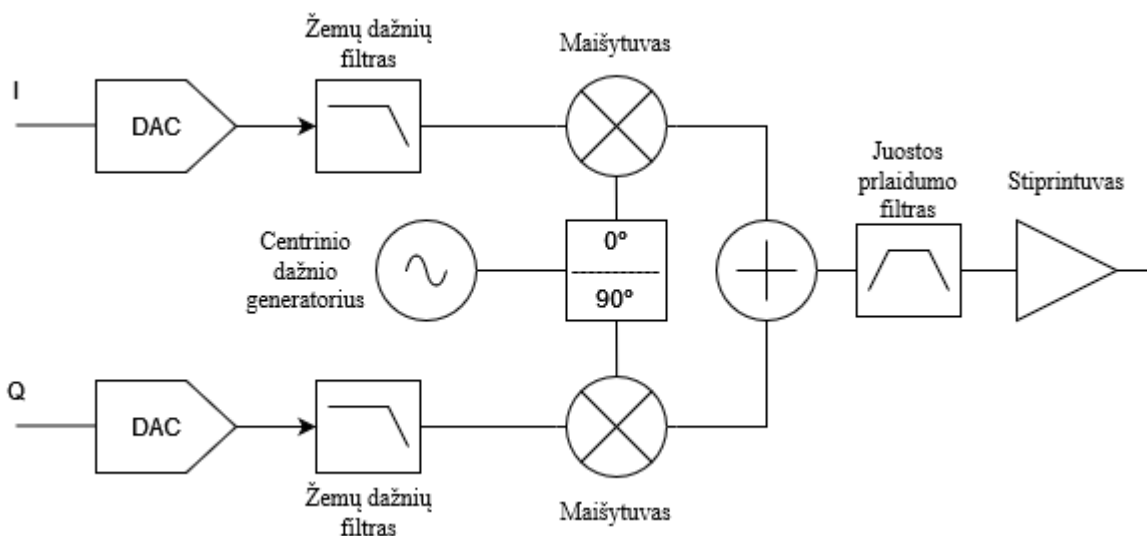
Darbo uždaviniai:

- išanalizuoti jau sukurtus siųstuvų ir bevielio ryšio technologijų identifikavimo būdus;
- skirtingų siųstuvų duomenų rinkimas ir analizė 868 MHz dažnio ruože;
- atlikti radijo signalų, paremto gilaus mokymosi, klasifikavimo algoritmų analizę;
- radijo signalų požymių išskyrimo analizė ir duomenų paruošimas tinklo mokymui;
- sudaryti radijo siųstuvų klasifikavimo metodą, naudojant gilųjį neuroninį tinklą;
- sukurto algoritmo ir tinklo įvertinimas bei analizė.

1. Literatūros apžvalga

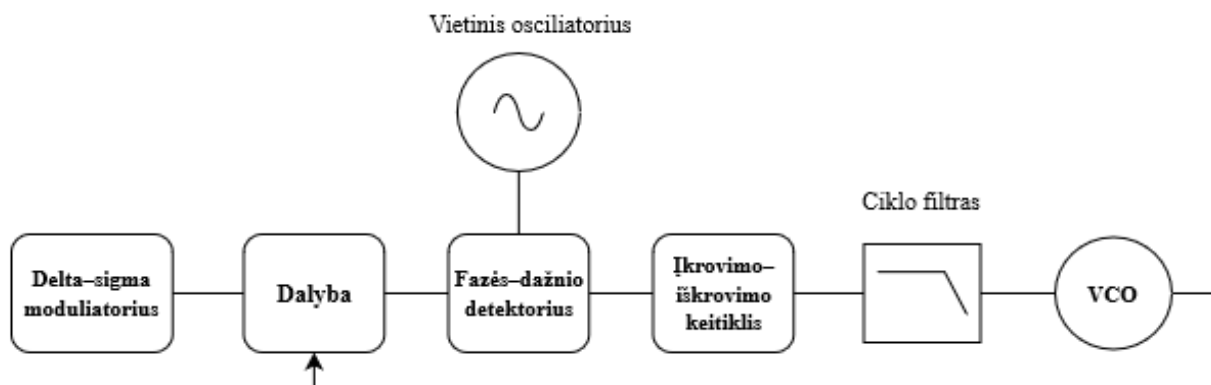
Bet koks radijo siųstuvas, dėl skirtingų gamybos vietų, skirtingos gamybos įrangos, tos pačios gamybos įrangos paklaidų, naudojamų komponentų įvairovės ir jų išsidėstymo spausdinto montažo plokštėje, susideda į unikalų elektromagnetinės kilmės parašą.

Dažniausiai naudojama technika yra heterodino signalo panaudojimas, kai bazinės juostos signalas yra pakeičiamas iš skaitmeninio į analoginį signalą pasinaudojant DAC keitiklį. Gautas analoginis signalas nufiltruojamas žemų dažnių filtru ir susumuojamas su sugeneruotu centriniu dažniu. Supaprastintas heterodino architektūros pavyzdys pateikiamas 1 pav.



1 pav. Supaprastinta heterodino siųstuvo blokinė diagrama

Dažniausiai naudojamas centrinio dažnio generatorius yra slenkančio kablelio – N (angl. *Fractional – N synthesizer*) sintezatorius (žr. 2 pav.). Šis signalo sintezatorius naudoja delta – sigma moduliatorių, kuris susideda iš keleto pagrindinių komponentų: fazės detektoriaus, ciklo filtro (angl. *Loop filter*), įkrovimo–iškrovimo keitiklio (angl. *Charge pump*), įtampa valdomo osciliatoriaus (angl. *Voltage controlled oscillator (VCO)*) ir programuojamo dažnio daliklio (angl. *Programmable frequency divisor*) [7][8].



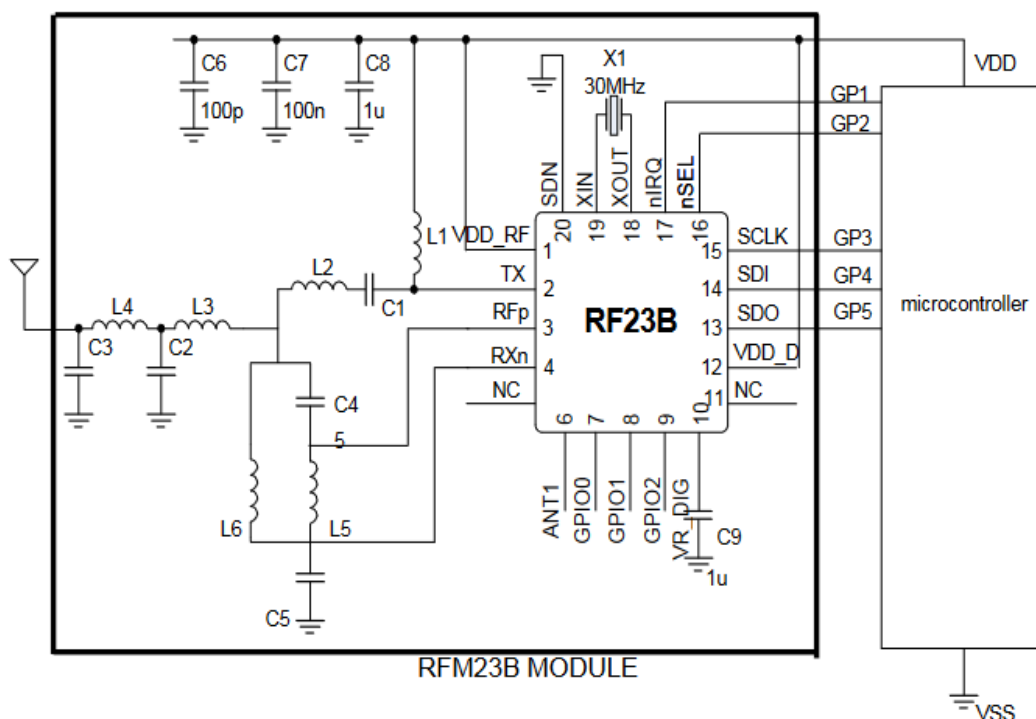
2 pav. Įtampa valdomo dažnio generatoriaus blokinė diagrama

Šios sistemos dalys suformuoja elektromagnetinį parašą, kuris nepriklauso nei nuo laiko, nei nuo duomenų tipo. Yra keturi pagrindiniai elementai, kurie daro įtaką siųstuvo stabilumui [9]: vietinio osciliatoriaus, fazės – dažnio detektoriaus, netiesinių efektų ir dedamieji dažniniai impulsai.

- **Vietinio osciliatoriaus triukšmas** (angl. *Local oscillator*) – trys pagrindiniai šaltiniai siuntimo grandinėje: DAC laikrodis, prie tarpinio dažnio kėliklio prijungtas LO ir VCO, kuris yra dažnio sintezavimo grandinėje;
- **Fazės–dažnio detektoriaus triukšmas** (angl. *Phase frequency detector (PFD)*) – tam tikri PFD virpesiai gali būti moduluojami į fazės pokyčius išėjime;
- **Netiesiniai efektai** (angl. *Nonlinear effects*) – įvairūs komponentų netiesiškumai iš PFD pirmiausiai patenka į stiprintuvą, kurie, kartu su švriu signalu yra netiesiškai sustiprinami. Šiems triukšmams pašalinti yra naudojamas slopinantis harmonikas, tam tikros pralaidumo juostos, filtras, tačiau dėl intermoduliacijos ir stiprintuvo iškraipymų maža dalis netiesinio triukšmo išlieka.

Skirtingai nei bevielio ryšio protokolai, kurie gali kisti su kiekviena nauja programinės įrangos iteracija, šie paminėti radijo siųstuvo triukšmų požymiai yra nekintantys ir jų imitacija yra sudėtinga.

Pateikiama pavyzdinė RFM22/23B modulio schema (žr. 3 pav.), kurioje vaizduojama, jog nors ir yra naudojamas RF23B lustas, tačiau jis yra sukonfigūruojamas pagal išorinius elektronikos komponentus, kurie, priklausomai nuo modulio modelio, gali atitinkamai skirtis – komponentų tipai ar jų vertės, o tokie skirtumai sudaro unikalų aparatinį parašą, kuris ir padeda atskirti šį modulį.



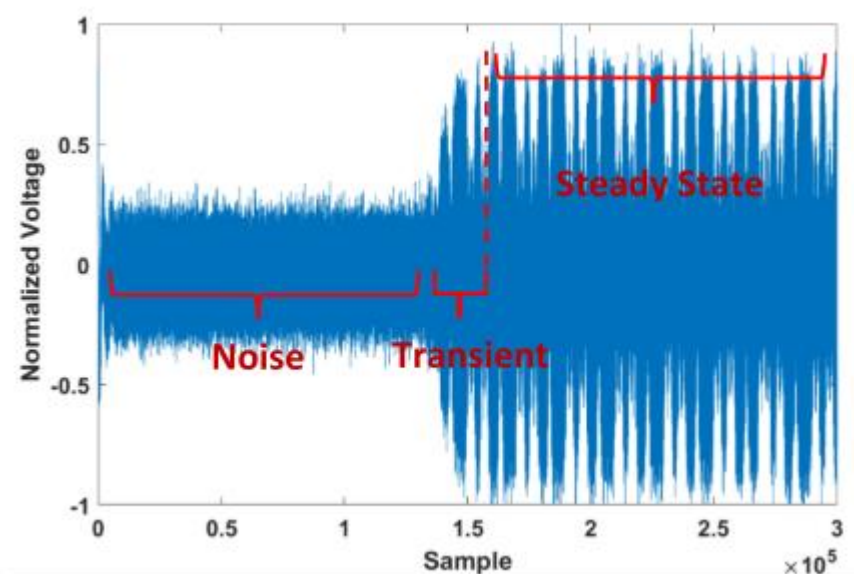
3 pav. RFM22/23B modulio schema [10]

1.1. Radijo siųstuvų klasifikavimo metodų apžvalga

Bevielio ryšio signalų spektre bet kokiame dažnyje radijo siųstuvo klasifikavimas gali būti išskiriamas į tradicinius ir gilioju mokymusi grįstus metodus. Naudojantis tradiciniais metodais, siųstuvo parametrus galima nustatyti pagal moduliaciją, statistinius parametrus, kylančio fronto ypatybes, laiko srities signalo kėlinius (angl. *Permutation*) ar dispersijos entropiją ir gauto signalo stiprumą.

Didelė dalis fizikinių reiškinių neturi konkretaus modelio ir yra natūraliai netiesiniai, tad tokios sistemos modelio aprašymas tampa ne tik sunkus, bet galbūt ir neįmanomas. Tokius reiškinius gana nesunku atskirti ir nustatyti pasinaudojant mašininį mokymąsi, kuris iš didelio duomenų kiekio gali išskirti netiesinius sistemos parametrus [11].

Radijo siųstuvų klasifikavimui dažniausiai naudojamos dvi signalo dalys: jo pereinamoji ir stabili dalis (žr. 4 pav.). Pereinamoji dalis yra signalo dalis, kai amplitudė pakyla iš triukšmo ribos iki jau siunčiamo signalo amplitudės, šios dalies požymiai ir yra labiausiai priklausantys nuo veikiančio siųstuvo. Pereinamasis signalas yra nano sekundžių ribose, tad jį pagauti gali būti sudėtinga. Kitu atveju, kai yra stabilus signalas, jo ištraukimas yra daug paprastesnis ir ši dalis yra statistiškai stabilesnė, tačiau, kaip prieš tai minėta, ji gali netikėtai kisti ir nuo įvairių programinės įrangos atnaujinimų.



4 pav. WiFi normalizuotas paketas su pažymėtais segmentais: triukšmas, signalo pereinamoji ir stabili dalis [12]

Lyginant įrenginių klasifikavimą naudojant pereinamąją ir stabiliąją signalo dalis ir turint po 50 skirtingų įrenginių klasių (duomenys buvo renkami iš skirtingų laivų siųstuvų), buvo sukurti gilaus mokymosi modeliai siųstuvo atpažinimui ir lyginamas tikslumas. Duomenys pateikiami 1 lentelėje. Iš duomenų matyti, jog beveik visais atvejais arba dažniausiai pereinamojo signalo panaudojimas yra pranašesnis nei stabilaus signalo naudojimas. Šio darbo eigoje bus nagrinėjamas siųstuvų klasifikavimas naudojant tik pereinamąjį signalą.

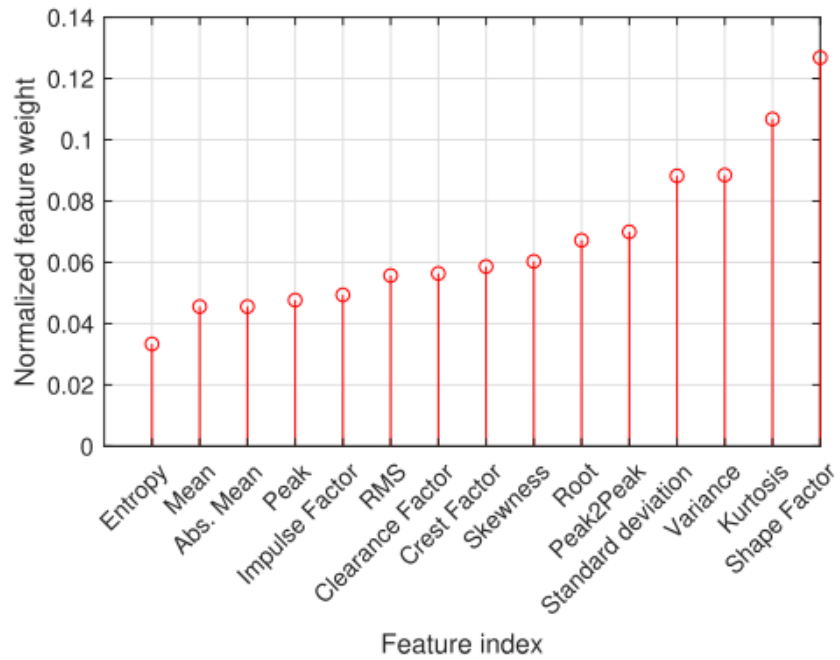
1 lentelė. Pereinamosios ir stabilios signalo dalies panaudojimo klasifikavime, tikslumo įvertinimas [13]

Modelis	Pereinamasis signalas	Stabilus signalas
	Tikslumas, %	
BiLSTM	87,95	26,73
ResNet-18	90,31	85,52
Vanilla Transformer	93,34	58,01
SwinT	94,34	76,85
ConvT	95,57	79,26
GLFormer	96,31	89,38

1.1.1. Mašininio mokymosi algoritmų panaudojimas

Keliuose nagrinėtuose darbuose yra naudojami signalo laiko – dažnio – energijos pasiskirstymo (angl. *Time–frequency–energy distribution (TFED)*) išskaičiuoti statistiniai požymiai: atskiri požymiai energijos – laiko ir energijos – dažnio srityse, pavyzdžiui, pereinamojo signalo dispersija ar distribucijų ekscesai. Abejuose darbuose TFED buvo skaičiuojamas naudojant Hilberto–Huango transformaciją, o gautieji išskaičiuoti požymiai buvo panaudoti atraminių vektorių klasifikatoriui (angl. *Support vector machine, SVM*) apmokyti. Viename darbe autoriai naudojo 8 telefonų duomenis 890–909 MHz dažnių juostoje, o duomenis rinko 500 Ms/s diskretizavimo dažniu, pralaidos juosta – 20 MHz. Iš viso naudojant 13 požymių, autoriams pavyko išgauti 100 % tikslumą klasifikuojant visus 8 telefonus. Kitame darbe, autoriai analizavo skirtingų gamintojų ir modelių telefonus - dviejų telefonų grupės su vienodais modeliais, tačiau skirtingais serijiniais numeriais. Analizuojami 20 - ies telefonų Bluetooth ryšio signalai, surinkti 20 Gs/s diskretizavimo dažniu. Autoriams pavyko išgauti 90,53 % tikslumą esant aukštam SNR (18–23 dB) ir naudojant tiesinį SVM klasifikatorių [14][15].

Radijo siųstuvų atpažinimas yra ypač naudingas ir aktualus bepiločiams dronams atpažinti, kuris vykdomas pasinaudojant dronų valdiklių siunčiamų signalų pereinamąją dalimi [16][17]. Požymiams ištraukti buvo panaudota trumpalaikė Furjė transformacija (STFT). Analizuojamų signalų centrinis dažnis 2.4 GHz, o signalai gauti osciloskopu 20 Gs/s diskretizavimo dažniu. Naudojama 17 dronų valdiklių. Atitinkamai, iš STFT gautos spektrogramos buvo skaičiuojami parametrai: vidurkis, standartinis nuokrypis, asimetrijos koeficientas, entropija, ekscesas, dispersija, piko reikšmė ir t.t. Pabrėžiamas teisingas pereinamojo signalo pradžios radimas. Dėl didelio požymių kiekio autoriai atliko kaimynystės komponentų analizę NCA (angl. *Neighborhood component analysis*) ir nustatė daugiausiai svorio turinčius požymius. Ekscesas, dispersija ir formos faktoriaus požymiai yra svarbiausi, nes mažesnis požymių kiekis sumažina tinklo persimokymo tikimybę. Minėtų požymių formulės pateikiamos 2 lentelėje.



5 pav. Autorių analizuojamų požymių vertė klasifikavimo procese naudojant mašininio mokymosi tinklą – SVM [17]

2 lentelė. Bluetooth signalo duomenų analizei panaudoti statistiniai metodai [17]

Požymis	Formulė	Matuoja
Vidurkis (μ)	$\frac{1}{N} \sum_{i=1}^N x_i$	Centrinę tendenciją
Absoliutus vidurkis (\bar{x})	$\frac{1}{N} \sum_{i=1}^N x_i $	Centrinę tendenciją
Standartinis nuokrypis (σ)	$\left[\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2 \right]^{\frac{1}{2}}$	Duomenų dispersiją
Asimetrijos koeficientas (γ)	$\frac{\sum_{i=1}^N (x_i - \bar{x})^3}{(N-1)\sigma_T^3}$	Asimetrija / formos aprašymas
Entropija (H)	$-\sum_{i=1}^N x_i \log_2 x_i$	Neapibrėžtumas
Kvadratų vidurkio kvadratinė šaknis (x_{rms})	$\left[\frac{1}{N} \sum_{i=1}^N x_i^2 \right]^{\frac{1}{2}}$	Amplitudė / vidutinė galia
Šaknis (x_r)	$\left[\frac{1}{N} \sum_{i=1}^N x_i ^{\frac{1}{2}} \right]^2$	Amplitudė
Ekscesas (k)	$\frac{\sum_{i=1}^N (x_i - \bar{x})^4}{(N-1)\sigma_T^4}$	„Uodegos“ / Formos aprašymas
Dispersija	$\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$	Duomenų dispersija
Viršūnės reikšmė (x_{pv})	$\max(x_i)$	Amplitudė
Viršūnė – apačia reikšmė (x_{ppv})	$\max(x_i) - \min(x_i)$	Bangos amplitudė

Formos faktorius (x_{sf})	$\frac{x_{rms}}{\bar{x}}$	Formos aprašymas
Viršūnės koeficientas	$\frac{x_{max}}{x_{rms}}$	Viršūnės kraštutinumumas
Impulso koeficientas	$\frac{x_{max}}{\bar{x}}$	Impulsas

Pasinaudojant minėtais statistiniais požymiais, buvo apmokyti SVM, k – arčiausių kaimynų algoritmas (kNN) ir atsitiktinių sprendimų medis, o jų rezultatai, esant 25 dB SNR, pateikiami lentelėje.

3 lentelė. Mašininio mokymosi algoritmų tikslumo palyginimas esant skirtingam valdiklių kiekiui [17]

Klasifikatorius	Tikslumas (esant 15 valdiklių)	Tikslumas (esant 17 valdiklių)
kNN	97,30	95,62
SVM	96,47	93,82
Random Forest (RandF)	98,53	96,32

Kai kurie autoriai savo darbuose pamini, jog pereinamojo signalo analizei naudojant STFT, Wigner–Ville ar Choi–Williams gali būti išgaunami nepakankamai geri požymiai, nes jie nėra adaptyvūs ir yra apriboti Heizenbergo principo. Svarstoma galimybė naudoti vilnelių metodą, tačiau prieš šio metodo panaudojimą yra privaloma iš anksto žinoti, kokios formos vilnelė bus naudojama. Taip pat, dėl vilnelės fiksuoto ilgio atsiranda energijos išsibarstymo tikimybė. Atsižvelgiant į šią analizę, autoriai nustatė, jog vilnelių metodas nėra pakankamai prisitaikantis prie duomenų. Autoriai nusprendė naudoti HHT metodą, pagal kurį galima paskaičiuoti TFED, o iš rezultatų apskaičiuoti statistinius parametrus. Duomenų rinkimui autoriai naudojo osciloskopą, kurio diskretizavimo dažnis buvo 500 Msps, o dažnių juostos plotis – 20 MHz. Pereinamieji signalai renkami iš mobiliųjų telefonų siųstuvų: keturi Nokia 5230, du Motorola Me525 ir du Xiaomi 1. Centrinis dažnis buvo 890 – 909 MHz. Turint HHT, autoriai skaičiavo bendrus požymius, taip pat požymius iš dažnių ašies, požymius iš laiko ašies ir bendrus dažnių – laiko ašies.

Bendrieji požymiai sudarė visų TFED reikšmių sumą, viso pereinamojo signalo trukmę bei trukmę nuo pradžios iki maksimalaus energijos taško vietos. Trukmė skaičiuojama imties taškais. Skaičiuojant dažnių ašies požymius, laiko – dažnio juosta padalinama į N vienodų segmentų dažnių ašyje, kurių energija tuomet žymima F_{En_i} ($i = 1, 2, \dots, N$). Vektorius $[F_{En_1}, F_{En_2}, \dots, F_{En_N}]$ žymi energijos pasiskirstymą dažnių ašyje. Naudojant šį vektorių, skaičiuojama energijos – dažnio pasiskirstymo entropija:

$$s(q) = - \sum_{i=1}^N q_i \ln(q_i) \quad (1)$$

čia $q_i = F_{En_i}/En_{sum}$, kur kiekvienas segmentas padalinamas iš bendros energijos sumos – atliekamas normalizavimas.

Sekantis požymis yra energijos–dažnio ekscesas:

$$F_k = \frac{E(F_{En_i} - \mu)^5}{\sigma^5} \quad (2)$$

čia μ – požymių vektoriaus vidurkis, o σ – standartinis nuokrypis. Toliau skaičiuojamas energijos – dažnio asimetrijos koeficientas:

$$F_{ske} = \frac{E(F_{En_i} - \mu^3)}{\sigma^3} \quad (3)$$

Paskutinis požymis pateikiamas kaip energijos – dažnio išsibarstymo centras:

$$F_{centre} = \sum_{i=1}^N i * \frac{F_{En_i}}{En_{sum}} \quad (4)$$

Lygiai tie patys parametrai yra apskaičiuojami ir laiko ašyje. Autoriai taip pat apjungia laiko ir dažnių ašis, iš kurių gaunami papildomi požymiai. Visas TFED padalijamas į $N_F * N_T$ kvadrato formos segmentus. Energija jau žymima kaip 2D matrica: $En_{ij} (i = 1, 2, \dots, N_F; j = 1, 2 \dots, N_T)$. Turint segmentuotus duomenis, skaičiuojama laiko – dažnio – energijos entropija:

$$s(q) = - \sum_{i=1}^{N_F} \sum_{j=1}^{N_T} i \times j \times \frac{En_{ij}}{En_{sum}} \quad (5)$$

Paskaičiuojamas ir laiko – dažnio – energijos pasiskirstymo centras:

$$T_{F_{centre}} = \sum_{i=1}^{N_F} \sum_{j=1}^{N_T} i \times j \times \frac{En_{ij}}{En_{sum}} \quad (6)$$

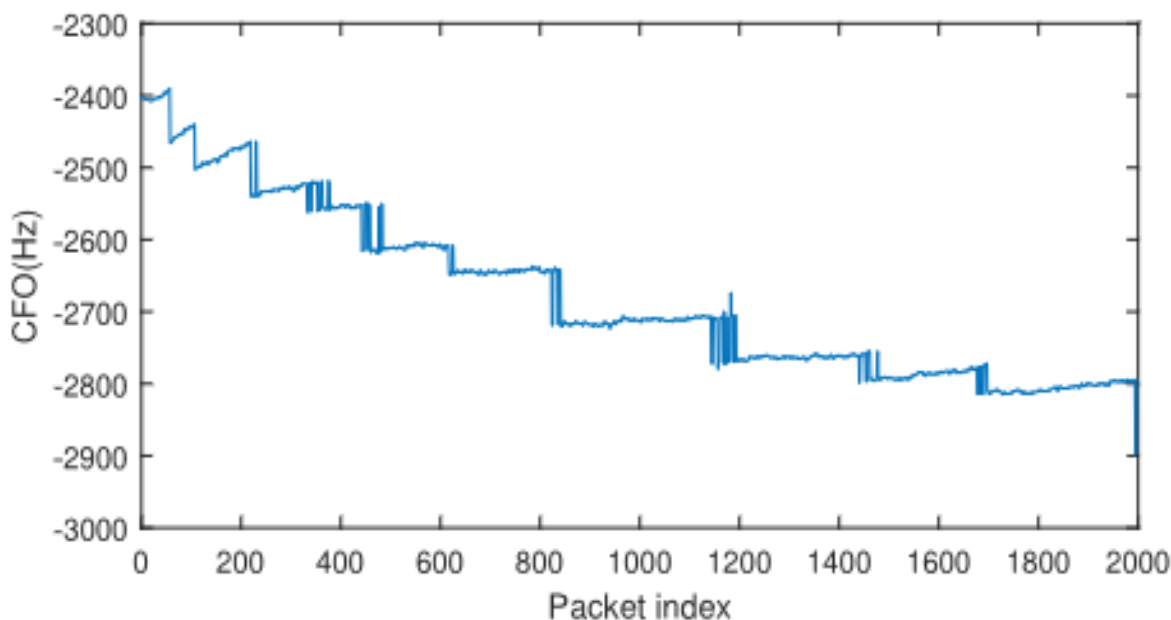
Panaudojus šiuos požymius, buvo panaudotas PCA metodas požymiams atrinkti bei apmokytas SVM tinklas. Autoriai gavo 100 % klasifikavimo tikslumą tarp 8 įrenginių.

Išanalizavus keletą klasikinių mašininio mokymosi algoritmų panaudojimus, matyti, jog yra naudojama daug statistinių požymių, kurie turi būti rankiniu būdu išskaičiuojami iš duomenų. Taip pat, toks būdas nėra tinkamas labai dideliems kiekiams klasifikuojamų objektų, nes didėjant kiekiui, panašėja statistiniai požymiai. Atsižvelgiant į pranašumus, šie modeliai gali būti greitai suprojektuojami bei apmokomi ir yra tinkami naudoti, jei klasifikuojamų objektų imtis yra nedidelė, nes, statistiniu požiūriu, didėjant imties dydžiui, didėja tikimybė, jog skirtingų RS išskaičiuoti požymiai taps neatskiriami.

1.1.2. Giliojo mokymosi algoritmų panaudojimas

Vienas iš didžiausių minusų konstruojant požymius minėtiems algoritmams yra tai, jog gauti rezultatai turi didelę priklausomybę nuo ištrauktų požymių kokybės, o tai reikalauja domeno eksperto žinių, taipogi, įrenginių aparatinės įrangos netobulumai gali būti susiję tarpusavyje, kuriuos rankiniu būdu atskirti gali būti labai sudėtinga ar net neįmanoma. Giliojo mokymosi sistemos yra pranašesnės, nes dėl neuroninio tinklo ypatumų, turint pakankamai duomenų, jos gali ištraukti paslėptus ir sudėtingus, aukštesnio lygio, požymius [18].

Dauguma darbų, kurie analizuoja LoRa įrenginių identifikavimą, naudoja vienokį ar kitokį neuroninį tinklą [4][18][19]. Vieni autoriai naudojo 20 įrenginių klasių, kurių duomenys buvo surinkti ties 868,1 MHz centriniu dažniu, o diskretizavimo dažnis – 1 MS/s. Klasifikavimui panaudotas CNN tinklas su centrinio dažnio dreifo kompensavimu, kurio įvestys buvo I/Q duomenys – tikslumas 83,36 %, Furjė transformacijos rezultatai – tikslumas 87,36 %, bei spektrograma, kurios tikslumas buvo aukščiausias ir siekė 96,44 % [18].



6 pav. Darbe nagrinėtų skirtingų įrenginių centrinio dažnio poslinkio vertės [18]

Kiti autoriai panaudojo skirtingų variacijų LSTM ir 1D/2D CNN tinklus su I/Q ir Furjė transformacijos įvestimis klasifikuojant skirtingą kiekį įrenginių: 10, 20, 50 ir 100 [36]. Naudojant LSTM tinklą, autoriai pastebėjo 12–47 % pablogėjimą padidinus įrenginių kiekį, tačiau naudojant CNN, klasifikavimo klaida sumažėjo iki 17 %. Taip pat, šiame darbe buvo panaudotas ir mokymosi duomenų dirbtinis išplėtimas naudojant sugeneruotus baigtinio impulso atsako filtrus, kurie simuliuoja įvairius atspindžių kanalus, šis metodas, geriausiu atveju, pagerino klasifikavimo tikslumą tarp 19 ir 36 %.

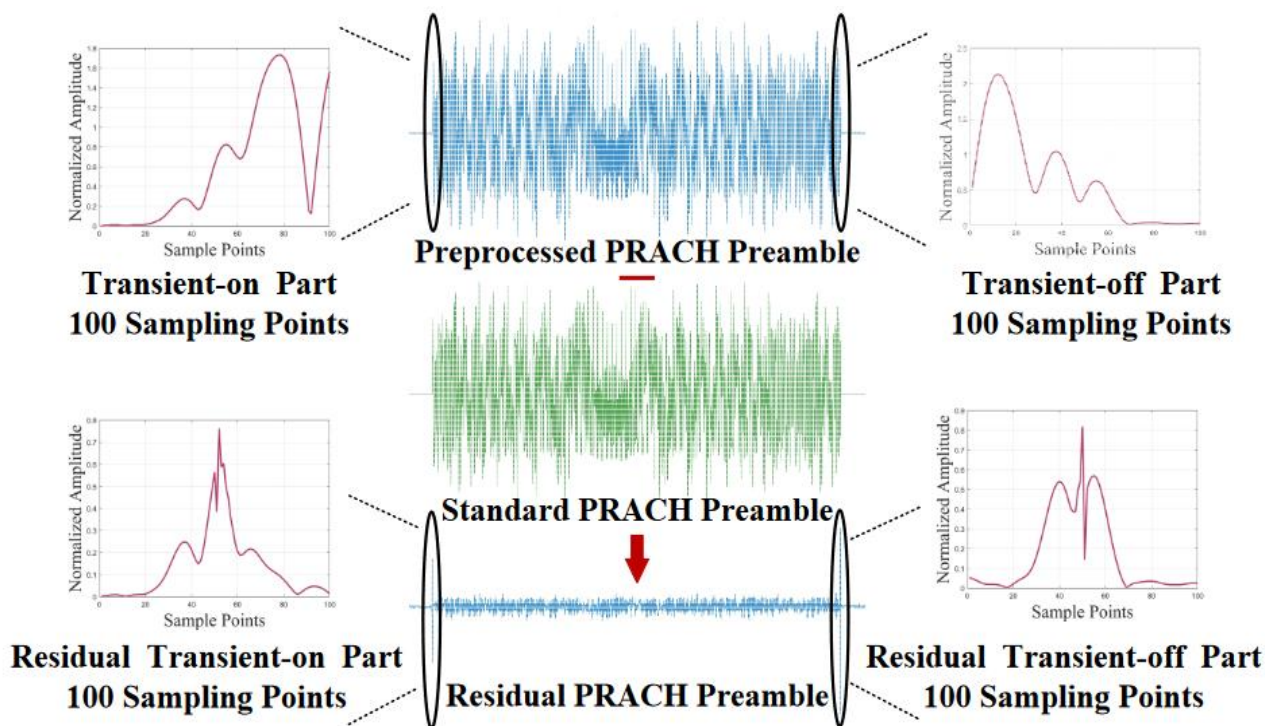
Literatūroje taip pat buvo atsižvelgta ir į gana svarbią CNN tinklų problemą - jie gali priimti tik fiksuoto dydžio įvestis, o tai reiškia, jog gali būti klasifikuojami tik vienodo ilgio signalai, nors realioje radijo bangų aplinkoje to tikėtis iš signalų negalima. Autoriai aiškina, jog tokia limitacija yra dėl paprasto neuronų sluoksnio (angl. *Dense layer*) [20]. Darbe yra naudojami modifikuotas CNN tinklas, LSTM, GRU (angl. *Gated recurrent unit*) ir transformerio modeliai. Taip pat naudojamas dirbtinis duomenų išplėtimas - naudojami AWGN kanalai ir išplečiamas duomenų SNR ruožas. Įvestims buvo panaudota autorių aprašyta ir įrodyta nuo kanalo nepriklausoma spektrograma. Modelis buvo testuojamas nuo 0 dB iki 40 dB SNR ruože ir, atitinkamai, gauti rezultatai pateikiami lentelėje. Rezultatai pateikiami 4 lentelėje– duomenų kiekis sudarė 90 000 paketų, iš kurių SF7, SF8 ir SF9 konfigūracijos turi po 30 000 paketų. Kiekvienam įrenginiui tenka po 9 000 paketų.

4 lentelė. Modelių tikslumas naudojant duomenų praplėtimą esant skirtingoms SNR reikšmėms klasifikuojant 10 skirtingų LoRa įrenginių naudojant SF7, SF8 ir SF9 LoRa konfigūracijas [20]

Modelis	Tikslumas (0 - 40 dB)	Parametrų skaičius
„Flatten–Free“ CNN	22,10–100 %	675 594
LSTM	24,02–99.98 %	856 586
GRU	23,82–99.96 %	644 618
Transformer	21,98–99.2 %	348 938

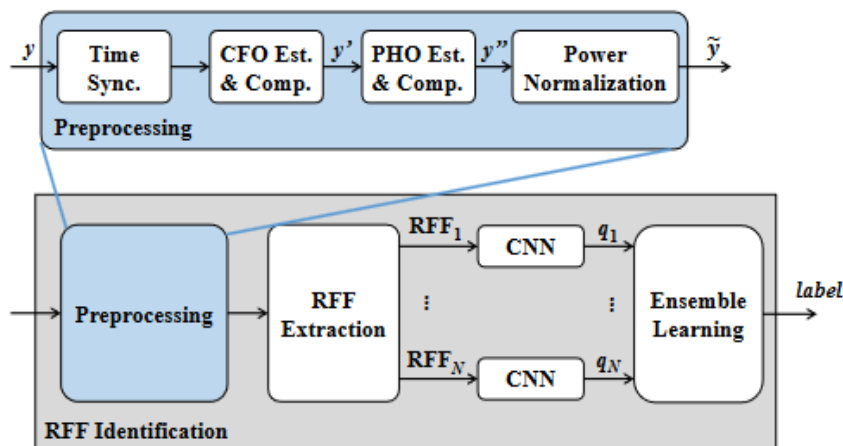
Matyti, jog rezultatai yra labai panašūs, tačiau yra didelis skirtumas tarp modeliuose esančių parametrų skaičiaus – transformerio modelis yra mažiausiai skaičiavimų reikalaujantis ir autorių apibūdinamas kaip paprasčiausias bei greičiausias [18][20].

Dar vienas bandymas panaudoti CNN tinklus yra klasifikuojant mobiliuosius telefonus, naudoti iš LTE komunikacijos sistemos ištrauktus PRACH (angl. *Physical random access channel*) preambulės pereinamuosius signalus esant skirtingoms eNB stotims [21]. PRACH preambulė yra pirmasis iš UE išsiunčiamas signalas, kai norima prisijungti prie eNB. PRACH preambulė naudojama sinchronizacijai su stotimi tam, kad būtų galima naudoti stoties resursus komunikacijai. Autoriai vietoj tiesioginio pereinamojo signalo ištraukimo, naudoja atliekamo signalo ištraukimą, t.y. – gauta PRACH preambulė sulyginama su idealia standartine ir yra gaunamas atliekamas signalas, kuris ir naudojamas klasifikavimui (žr. 7 pav. Požymių ištraukimo proceso vizualizacija [21]).



7 pav. Požymių ištraukimo proceso vizualizacija [21]

Gautos preambulės apdorojimui atliekama sinchronizacija, CFO (angl. *Carrier frequency estimation*) ir PHO (angl. *Phase offset estimation*) ištaisymai bei normalizavimas. Darbe naudoti šeši fiziniai eNB įrenginiai ir penki mobilieji telefonai: Google Nexus 5, XIAOMI MCT3B, HUAWEI P9, Google Nexus 6P ir HONOR 30 Lite. Duomenys renkami naudojant USRP B205 programuojama radija.



8 pav. Įrenginio identifikavimas naudojant ištrauktus požymius [21]

Autoriai išskyrė keletą požymių: preambulės pradžios ir pabaigos viso pereinamojo signalo segmentą, pereinamojo signalo liekaną, stabilaus signalo liekaną (be pereinamojo signalo) ir viso signalo liekaną lyginant su idealiu signalu. Kiekvienas iš šių požymių buvo panaudotas apmokant po atskirą CNN tinklą. Apmokymui surinkta po 200 preambulių per telefoną, o testavimui po 60. Klasifikuojant mobiliuosius telefonus naudojant tik vieną eNB, buvo pasiektas geriausias tikslumas su preambulės pabaigos pereinamojo signalo liekana – 96 %, tačiau naudojant visus šešis požymius, gaunamas net 99 % tikslumas. Naudojant visus skirtingus eNB, geriausias požymis išlieka tas pats ir gaunamas ~74 % tikslumas, tačiau, šiuo atveju, geriausias tikslumas pasiekiamas naudojant preambulės signalo pradžios ir pabaigos pereinamojų dalių liekanas – 83 %. Mobiliojo telefono klasifikavimo diagrama pateikiama 8 pav., iš kurios matyti, jog yra atliekama pilna signalo sinchronizacija, galios normalizacija ir požymių išskaičiavimas. Išskaičiuotus požymius, jie paduodami į CNN tinklus, kurių rezultatai apjungiami per ansamblinį mokymąsi.

1.2. Giliųjų neuroninių tinklų radijo siūstuvų klasifikavimo požymiai

Aukšto tikslumo radijo siūstuvų klasifikavimas turi didelę priklausomybę nuo teisingų požymių parinkimo, o ir siūstuvų identifikavimas turi kitokias charakteristikas nei vaizdų ar natūralios kalbos apdorojimo (angl. *Natural Language Processing*) sritys. Radijo signalai turi unikalius požymius tiek laiko, tiek dažnių srityse, kurie yra labai priklausomi nuo naudojamos įrangos (jos komponentų) ir naudojamo protokolo. Jie taip pat turi ir ciklostacionarumo savybę [11], kuri atsiranda dėl pasikartojančios struktūros: preambulės, fizinio sluoksnio antraštės, ciklinio priešdėlio ir t.t. Minėta savybė naudojama tik stabilaus signalo fragmento požymiams išskirti.

Laiko sritis susideda iš I/Q duomenų, amplitudės ir fazės vektorių. Dažnių sritis turi daugiau signalą apibūdinančių naudingų parametrų: signalo plotį, centrinį dažnį ir galios spektrinį tankį (PSD). Dažnių sritis gali būti išskaičiuojama iš laiko srities panaudojant greitąją Furjė ar vilnelių transformacijas.

Signalų identifikavimo sėkmę neuroniniuose tinkluose nusako įvesties parametrai, kurie tiesiogiai koreliuoja su modelio tikslumu bei greitimeika. Su signalo parametrais, pavyzdžiui, I/Q duomenys, kurie yra išgaunami iš neapdoroto ir netransformuoto radijo signalo, galima sukurti primityvų klasifikatorių, tačiau su tam tikrais ribojimais – jautrumas ir priklausomybė nuo kanalo kokybės, triukšmo, mažas panašių protokolų atskiriamumas, ilgesnis apmokymo laikas ir mažas panašios komponentų sudėties ir gamybos įrenginių atskiriamumas [11].

1.3. Pereinamojo signalo pradžios nustatymas

Norint iš pereinamojo signalo išskirti bet kokius požymius, pirmiausiai yra reikalingas teisingas pereinamojo signalo pradžios ir galo nustatymas. Literatūroje yra minima keletas pereinamojo signalo pradžios nustatymo algoritmų: vidurkio pokyčio detektavimas MCPD (angl. *Mean change point detection*), dispersijos fraktalinio matmens slenksčio detektavimas VFDTD (angl. *Variance fractal dimension threshold detection*), Bajeso žingsnio kitimo detektavimas BSCD (angl. *Bayesian step change detection*), fazės kitimo detektavimas PD (angl. *Phase detection*) ir energijos kriterijaus metodas naudojant fazės arba amplitudės kitimo duomenis EC (angl. *Energy criterion*) [12][22][23].

5 lentelė. Pereinamojo signalo pradžios metodų palyginimas [22][23]

Metodas	Sudėtingumas	Tikslumas, %		Skaičiavimo trukmė, s	
MCPD	$O(n)$	91,6–98,5	90,0–99,0	211,61	1,03
VFDTD	$O(n^2)$	96,8–98,8	95,0–99,0	3,33	2,31
BSCD	$O(n^3)$	87,0–90,9	65,0–78,0	447,95	39,35
PD	$O(n)$	94,0–97,9	95,0–97,0	0,19	0,25
EC- α	$O(n)$	99,2–99,3	98,0–99,0	0,05	0,01

Taip pat, pateikiami metodų privalumai ir trūkumai – išanalizavus 5 lentelė ir 6 lentelė lentelių duomenis, matyti, jog dėl skaičiavimo trukmės, paprastumo ir tikslumo, optimaliausias algoritmas yra EC- α , tačiau reikia atsižvelgti ir į tai, kokia yra signalo aplinka ir koks gauto signalo SNR.

6 lentelė. Pereinamojo signalo pradžios metodų privalumai ir trūkumai [22][23]

Metodas	Privalumai	Trūkumai
MCPD	Gana geras detektavimas; Paprastas; Nereikia slenksčio;	Prastėja gebėjimas atpažinti mažėjant SNR; Skaičiavimo trukmė yra vidutinė;
VFDTD	Labai gerai detektavimas; Gana stabilių rezultatų metodas;	Sudėtingas apskaičiavimas; Naudoja slenksčio reikšmę;
BSCD	Nenaudoja slenksčio; Geras aštraus pereinamojo signalo detektavimas;	Gana žemas detektavimo tikslumas; Atliekami labai sudėtingi skaičiavimai;

PD	Labai geras detektavimas; Paprasta implementacija;	Naudoja slenksčio reikšmę; Mažėja detektavimo tikslumas mažėjant SNR;
EC- α	Labai geras detektavimas; Paprasta implementacija; Nenaudoja slenksčio;	Jautrus parinktai θ reikšmei - skirtingos reikšmės prie skirtingų SNR; Menka degradacija, jei signalas turi labai aštrią pereinamojo signalo charakteristiką;

Dėl realizavimo paprastumo ir greitaveikos, šiame tyrime buvo pasirinktas EC- α metodas [24][25]. Originalus metodas vadinamas Hinklio (angl. *Hinkley*) kriterijaus metodu, kuris buvo sukurtas dar 1971 metais. Dėl metodo panaudojimo naujumo šioje srityje iš anksto nebuvo žinoma, ar gauti rezultatai bus tinkami darbui, tačiau atlikus preliminarius testus ir analizę, buvo gauti patenkinami rezultatai. Šis metodas detektuoja signalo pradžią pagal signalo energijos pokytį. Pirmiausiai apskaičiuojama signalo kumuliacinė energija:

$$E[i] = \sum_{k=0}^i \left(\sqrt{I(k)^2 + Q(k)^2} \right), i = 1, \dots, N, \quad (7)$$

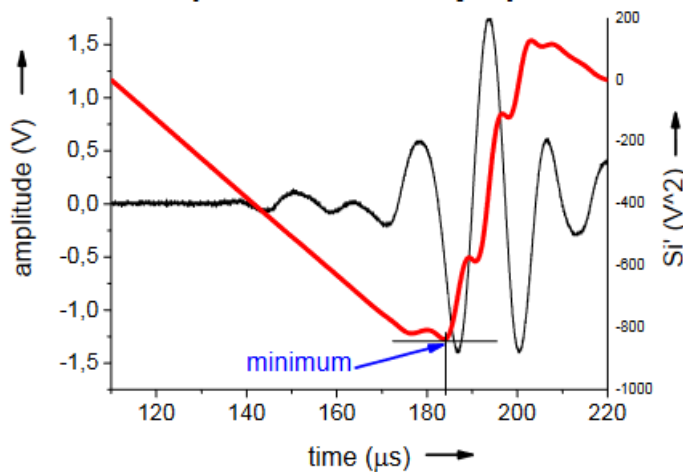
čia N – signalo ilgis. Tuomet, signalas yra atskiriamas nuo triukšmo pagal energijos kreivės formulę:

$$E'[i] = E[i] - i\delta \quad (8)$$

čia i – iteracija, δ – neigiama kitimo tendencija ir aprašoma:

$$\delta = \frac{E_N}{\theta * N} \quad (9)$$

čia E_N – viso signalo bendra energija, o θ – faktorius, nuo kurio priklauso, kaip greitai kinta neigiama tendencija. Atsižvelgiant į formules, matyti, jog apskaičiuotos energijos kreivės globalus minimumas bus analizuojamo signalo pradžia. Algoritmo veikimo pavyzdys pateikiamas 9 pav., kuriame ir matomas signalo energijos sumažėjimas iki globalaus minimumo, kuris ir yra impulso pradžios taškas.



9 pav. Akustinio signalo (plona linija) su jo energijos kreive (stora linija) minimumas [24]

1.4. Pereinamojo signalo kompleksinė analizė

Radijo signalų apdorojimas eina išvien su kompleksinėmis reikšmėmis ir jų apdorojimu, tad ir analizėje yra būtina į tai atsižvelgti, nes naudojant tik realią arba menamą dalį ar iš jų išskaičiuotą tik amplitudę ar fazę, gali būti prarandama svarbi informacija, kuri sieja amplitudę su signalo faze. Hilberto–Huango transformacijos analizės metodas yra paremtas daugiamačio empirinio režimo EMD (angl. *Empirical mode decomposition*) panaudojimu. EMD metodas išskaido pagrindinį signalą į fiksuoto kiekio vidinio režimo funkcijas IMF (angl. *Intrinsic mode functions*), kurios yra tiesiniai ir stacionarūs signalai, iš kurių susideda analizuojamas signalas. Originalus signalas gaunamas sudedant visus IMF signalus:

$$x[t] = \sum_{k=1}^K d_k[n] + r[n] \quad (10)$$

čia $x[t]$ – daugiakomponentis signalas, $d_k[n]$ – M -oji vidinio režimo funkcija, o $r[n]$ – vidinio režimo liekana. Tuomet, kiekvienam IMF signalui panaudojama Hilberto transformacija, kurią taikant, apskaičiuojamos signalų momentinės charakteristikos: momentinis dažnis, momentinė amplitudė ir momentinė fazė. Turint šias charakteristikas, generuojamas Hilberto spektras, kuris ir yra signalo laiko - dažnio - energijos pasiskirstymas. Klasikinė Hilberto transformacija yra skirta tik realios signalo dalies apdorojimui, tačiau šio darbo atveju yra naudojamas kompleksinis signalas, kuris susideda iš realios ir menamos dalių. Atitinkamai, pasinaudojant Toshihisa Tanaka sukurtu algoritmu, galima gana paprastai kompleksinį signalą konvertuoti tik į realų signalą. Algoritmo idėja remiasi tuo, jog neigiamos dažnių komponentės yra perkeliamos į teigiamas dažnių komponentes pasinaudojant idealiu juostos filtru bei kompleksiniu transponavimu.

$$H(e^{j\omega}) = \begin{cases} 1, & 0 \leq \omega < \pi \\ 0, & -\pi \leq \omega < 0 \end{cases} \quad (11)$$

$$X_+(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}) \quad (12)$$

$$X_-(e^{j\omega}) = H(e^{j\omega})X^*(e^{-j\omega}) \quad (13)$$

čia $X^*(e^{-j\omega})$ yra transponuota funkcija $X(e^{j\omega})$. Atitinkamai, pasinaudojant Hilberto transformacijos taisyklėmis ir atvirkštine Furjė transformacija, gauname:

$$x_+[n] = \text{Real}(F^{-1}[X_+(e^{j\omega})]) \quad (14)$$

$$x_-[n] = \text{Real}(F^{-1}[X_-(e^{j\omega})]) \quad (15)$$

Gautieji vektoriai $x_+[n]$ ir $x_-[n]$ yra tik realios reikšmės, tad galima pritaikyti įprastinį EMD metodą, kuris išskaido signalą į IMF dedamąsias:

$$x_+[n] = \sum_{i=1}^{N_+} x_i[n] + r_+[n] \quad (16)$$

$$x_-[n] = \sum_{i=-N_-}^{-1} x_i[n] + r_-[n] \quad (17)$$

Gauto signalo rekonstrukcija į vieną bendrą kompleksinį signalą išreiškiama:

$$x[n] = (x_+[n] + jH[x_+[n]]) + (x_-[n] + jH[x_-[n]])^* \quad (18)$$

čia H – Hilberto transformacija.

1.5. Literatūros analizės apibendrinimas

Atlikus RS klasifikavimo metodų analizę, buvo pastebėta, jog yra daug ir įvairių būdų signalo ir įrenginių klasifikavimui naudojant mašininį mokymąsi ar giliuosius neuroninius tinklus. Dauguma autorių duomenų rinkimui naudoja didelius diskretizavimo dažnius 20 Ms/s, 500 Ms/s ar net 20 Gs/s ir toks diskretizavimo dažnių panaudojimas sudaro milžiniškus duomenų kiekius, ko pasekoje, sudėtingėja ir ilgėja tinklų mokymas, o tai apsunkina tokias sistemas naudoti efektyviai. Taip pat, pastebėta, jog giluminių neuroninių tinklų panaudojimas yra efektyvesnė priemonė nei mašininio mokymosi, nes tuomet nebereikia atlikti sudėtingų skaičiavimų bei nėra būtina turėti specialisto žinių – giluminiai neuroniniai tinklai patys apsimoko ir atranda daug sudėtingesnius parametrus, be to, didėjant technologijų kiekiui, yra universalesni nei mašininio mokymosi algoritmai, tačiau yra labai svarbus duomenų apdorojimas prieš paduodant juos mokymuisi. Visuose darbuose, kur naudojamas pereinamasis signalas, yra pabrėžiamas šios signalo dalies pradžios ir pabaigos radimo svarbumas. Nagrinėtuose darbuose dominavo BSCD metodas, tačiau gana neseniai buvo pateiktas ir energijos kriterijaus metodas EC- α , kuris savo skaičiavimo paprastumu ir tikslumu turi pranašumą pereinamojo signalo pradžios detektavime.

2. Duomenų rinkimo architektūra ir įranga

2.1. Darbe naudojama įranga

2.1.1. Duomenų rinkimui naudojama įranga


Didžioji dalis nagrinėtų darbų duomenų rinkimui naudoja osciloskopus su maišytuvais ir dideliais diskretizavimo dažniais [14][15], tačiau šiame darbe signalų įrašymui buvo panaudota Analog Devices įmonės kurta, programuojama radija Pluto SDR (žr. 10 pav.), kuri palengvina signalų apdorojimo procesą atsisakant maišytuvų ir kurių išvestis yra kompleksiniai duomenys. Ši radija turi 12 bitų raiškos kodas - analogas ir analogas - kodas keitiklius, centrinio dažnio ruožas gali būti reguliuojamas nuo 325 MHz iki 3.8 GHz, o maksimali duomenų pralaidos juosta yra 20 MHz. Pačios radijos generuojamas triukšmas imtuvo pusėje yra iki 3.5 dB. Radijos viduje naudojamas AD9363 siųstuvus – imtuvas, kurio įvestys ir išvestys sujungtos su Xilinx® Zynq Z-7010 programuojama logine matrica. Sąsajai su kompiuteriu naudojama USB 2.0 jungtis.



10 pav. Pluto SDR siųstuvus – imtuvas

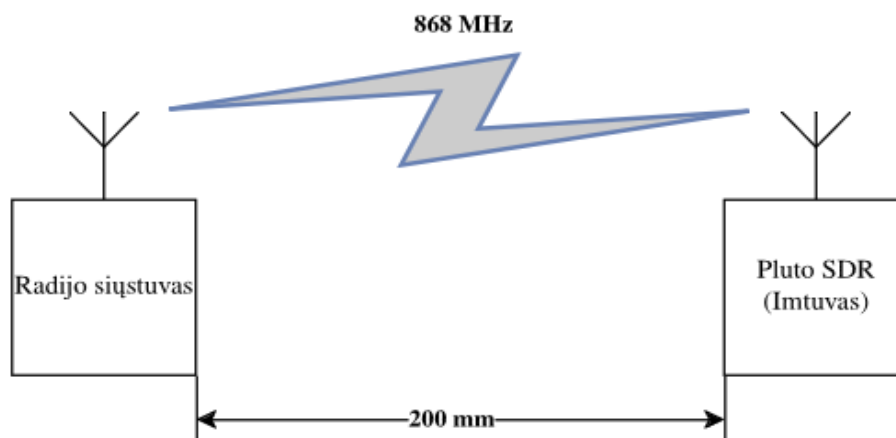
Duomenų rinkimui buvo pasirinktas 868 MHz centrinis dažnis ir prie kiekvieno siųstuvo prilituojama įvairiakryptė (angl. *Omni directional*) varinė, ketvirtadalio bangos (8,63 cm), savadarbė antena [26]. Imtuvui buvo panaudota 2 dBi įvairiakryptė antena, kurios modelis – JCG401. Ji optimizuota 824–894 ir 1710–2170 MHz dažnių diapazone, tad yra tinkama naudoti šiame tyrime.

7 lentelė. JCG401 antenos parametrai¹

Optimizuotos dažnių juostos, MHz	824 ~ 894 / 1710 ~ 2170	
Stiprinimas, dBi	2	
Poliarizacija	Tiesinė	
Impedansas, Ω	50	
VSWR	2,5 : 1	
Jungtis	SMA Male	
Aukštis, mm	50,2 ± 1,5	
Plotis, mm	18,6 ± 1	

¹ <https://wiki.analog.com/media/university/tools/pluto/users/jcg401.pdf>

Atstumas tarp siųstuvo ir imtuvo visada išlaikomas 20 cm, o tai yra pusė 868 MHz dažnio bangos ilgio. Toks atstumas pasirinktas norint išvengti imtuvo analogas - kodas keitiklio išotinio bei sumažinti tikimybę, jog per arti esančios antenos gali iškraipyti pereinamųjų signalų požymius dėl artimo elektromagnetinio lauko sąveikos. Aukšto lygio blokinė schema pateikiama 11 pav.



11 pav. Duomenų rinkimo sistemos supaprastinta blokinė schema.

2.1.2. Duomenų apdorojimui naudojama įranga

Duomenims apdoroti ir neuroninių tinklų apmokymui naudojamas stacionarus kompiuteris, kurio parametrai pateikiami žemiau:

- CPU: AMD Ryzen R9 7900X;
- GPU: Nvidia RTX 4090;
- RAM: DDR5 32GB;
- SSD: Samsung 1000 GB.
- OS: Arch Linux (linux-lts 6.6.21-1);
- Aplinka: Jupyter Notebook:
 - Python 3.10;
 - NumPy (1.26.4)²;
 - PyTorch (2.2.1)³;
 - Emd (0.6.2)⁴.

Visas darbas atliktas naudojantis „Jupyter Notebook“ programavimo aplinka ir *Python* kalbos branduolį.

² <https://pypi.org/project/numpy/>

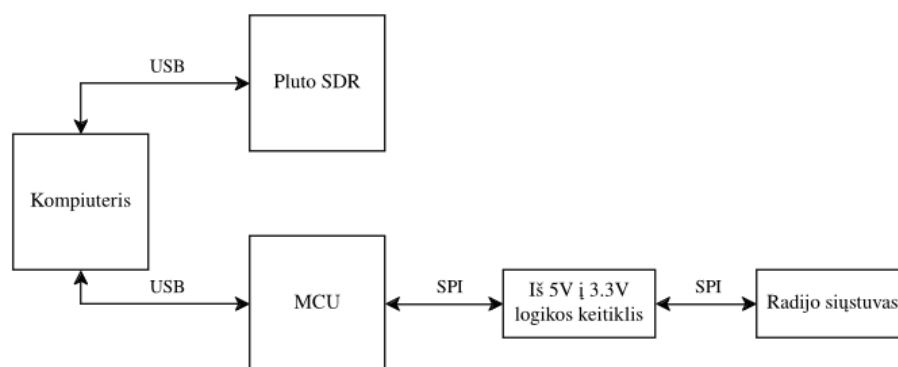
³ <https://pypi.org/project/torch/>

⁴ <https://pypi.org/project/emd/>

2.2. Duomenų rinkimo sistemos aprašymas

Dauguma bevielio ryšio signalų galima simuliuoti pasinaudojant MATLAB papildomais įrankiais: Signal Processing Toolbox⁵ ir WLAN Toolbox⁶, tačiau dėl šio darbo tyrimo specifikos, imituojami signalai nėra tinkami, nes jų pereinamieji signalai nebus unikalūs ir jie neatstos realaus radijo siųstuvo gamybos ypatumų. Dėl šios priežasties duomenų rinkimas buvo atliekamas pasinaudojant sukonstruota sistema, kurios pagrindinis tikslas - išsiųsti tam tikrą kiekį duomenų paketų ir juos priimti su radijo imtuvu. Po visų moduliacijų išsiuntimo parenkamas vis kitas radijo siųstuvas, kuris gali turėti skirtingas moduliacijas nei prieš tai buvęs siųstuvas.

Detalesnė struktūrinė schema pateikiama 12 pav. Prie kompiuterio per USB sąsają yra prijungiamas Pluto SDR ir mikrovaldiklis – naudojama plokštė yra Arduino Mega 2560 dėl atviro kodo bibliotekų, kurios reikalingos siųstuvams. Visi duomenų rinkime naudojami siųstuvai veikia 3.3V įtampos logika, tačiau Arduino Mega 2560 veikia tik su 5,5V įtampos logika, tad tarp SPI sąsajos yra reikalingas logikos keitiklis. Siųstuvams programuoti buvo pasirinktos Arduino bibliotekos: JeeLib⁷ ir RadioHead⁸ atviro kodo bibliotekos.



12 pav. Pilna duomenų rinkimo sistemos schema

Atliekant kiekvieną eterio įrašymą, Pluto SDR yra konfigūruojama pagal pateiktus lentelės duomenis. Imtuvo automatinis stiprinimas yra išjungiamas AGC (angl. *Automatic gain control*)) ir nustatoma statinė reikšmė.

8 lentelė. Pluto SDR imtuvo nustatyti pagrindiniai parametrai duomenų rinkimui

Centrinis dažnis	Diskretizavimo dažnis	Pralaidos juostos filtro plotis	Imtuvo stiprinimas
868 MHz	3 MHz	3 MHz	5 dB

Imtuvo diskretizavimo ir pralaidos juostos filtro pločio įtaka siųstuvų klasifikavimui pagal pereinamąjį signalą buvo nagrinėjama moksliniuose darbuose [27]. Šiame darbe autoriai naudojo IEEE 802.11b WiFi siųstuvų įrašytus pereinamuosius signalus, kurių trukmė apie 200 ns, o signalo plotis - 22 MHz. Originalus diskretizavimo dažnis buvo 5 GS/s. Naudoti diskretizavimo dažniai: 2.5

⁵ <https://www.mathworks.com/products/signal.html>

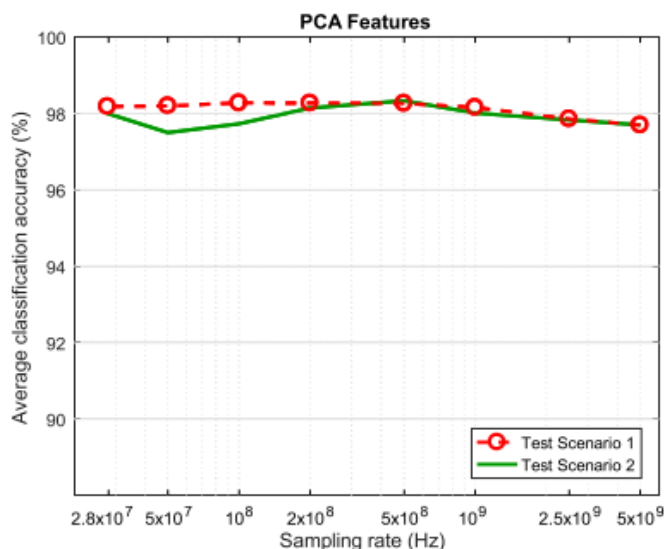
⁶ <https://www.mathworks.com/products/wlan.html>

⁷ <https://github.com/jeelabs/jeelib>

⁸ <https://www.airspayce.com/mikem/arduino/RadioHead/>

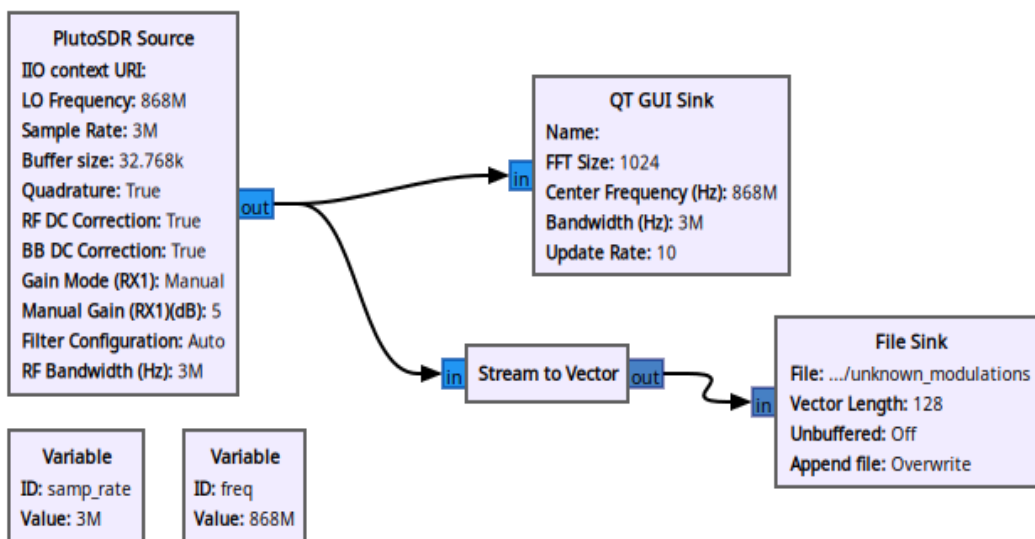
GS/s, 1 GS/s, 500 MS/s, 200 MS/s, 100 MS/s, 50 MS/s ir 28 MS/s. Autoriai pateikė rezultatus, jog naudojant PCA požymių sumažinimą (žr. 13 pav.), klasifikacijos rezultatai, naudojant skirtingus diskretizavimo dažnius, išlieka ties 98 %, o tai įrodo, jog norint išgauti gerus rezultatus, nėra būtina naudoti labai aukšto diskretizavimo dažnio, kuris gali prailginti įvairius skaičiavimus.

Atsižvelgiant į minėtą darbą, šiame darbe aukščiausia dažnių komponentė gali siekti iki 1 MHz, tad 3 MHz diskretizavimo dažnio turi pakakti geram klasifikacijos rezultatui gauti.



13 pav. Vidutiniai PCA ištrauktų požymių klasifikavimo rezultatai esant skirtingiems diskretizavimo dažniams [27]

Imtuvus konfigūruojamas ir duomenys renkami naudojant „GNU Radio“ programinę įrangą. „GNU Radio“ programine įranga sukurta blokų diagrama pateikiama 14 pav.






14 pav. „GNU Radio“ programinės įrangos blokų diagrama Pluto SDR valdymui (duomenų įrašymui)

2.3. Eksperimente naudoti radijo siųstuvai

Analizei atlikti buvo pasirinkti gana pigūs siųstuvų variantai – prie kiekvieno RS buvo prilituojami laidai ir antena, RS prijungiamas prie maketavimo plokštės. Siųstuvai pirkti iš UAB „Lemona“ parduotuvės. Žvelgiant vien į pačių siųstuvų vizualizacijas, matyti, jog jų visų komponentų išsidėstymas yra skirtingas.

9 lentelė. Darbe naudotų radijo siųstuvų techninės charakteristikos

Modelis	Specifikacijos	Nuotrauka
RFM95W ⁹	<p>Moduliacijos – FSK, GFSK, MSK, GMSK, LoRa ir OOK</p> <p>Plotis – nuo 7.8 kHz iki 500 kHz</p> <p>Skaidos faktorius – nuo SF6 iki SF12</p> <p>Palaikomi dažniai – 868/915 MHz</p> <p>Spinduliuojama galia – nuo 2 iki 17 dBm</p> <p>Veikimo įtampa – 3.3 V</p> <p>Veikimo srovė siunčiant:</p> <ul style="list-style-type: none"> • 20 dBm – 120 mA; • 13 dBm – 29 mA; • 7 dBm – 20 mA; <p>Kaina – 12 €.</p>	
RFM22B ¹⁰	<p>Moduliacijos – FSK, GFSK ir OOK</p> <p>Plotis – nuo 2.6 kHz iki 620 kHz</p> <p>Palaikomi dažniai – 433/470/868/915 MHz</p> <p>Spinduliuojama galia – nuo 1 iki 20 dBm</p> <p>Veikimo įtampa – nuo 1.8 iki 3.3 V</p> <p>Veikimo srovė siunčiant:</p> <ul style="list-style-type: none"> • 20 dBm – 85 mA; • 13 dBm – 30 mA; • 1 dBm – 18mA; <p>Kaina – 10.80 €.</p>	
RFM69HW ¹¹	<p>Moduliacijos – FSK, GFSK, MSK, GMSK ir OOK</p> <p>Plotis – nuo 2.6 kHz iki 500 kHz</p> <p>Palaikomi dažniai – 315/433/868/915 MHz</p> <p>Spinduliuojama galia – nuo -1 iki 20 dBm</p> <p>Veikimo įtampa – nuo 1.8 iki 3.3 V</p> <p>Veikimo srovė siunčiant:</p> <ul style="list-style-type: none"> • 20 dBm – 130 mA; • 13 dBm – 45 mA; • 0 dBm – 20 mA; • -1 dBm – 16 mA; <p>Kaina – 12.60 €.</p>	

⁹ https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf

¹⁰ <https://www.sparkfun.com/datasheets/Wireless/General/RFM22B.pdf>

¹¹ <https://datasheetspdf.com/pdf-file/748015/HOPERF/RFM69HW/1>

3. Surinktų radijo siųstuvų duomenų analizė

Prieš kuriant RS klasifikavimo modelį, reikia išanalizuoti surinktus skirtingų siųstuvų signalus ir iš jų išskirti pereinamuosius signalus. Analizuojami RS turi didelį skaičių moduliacijos, pločio ir dažnio nuokrypio kombinacijų variantų, tad stengiantis limituoti duomenų kiekį, buvo pasirinktos tik tam tikros moduliacijos rūšys – jei siųstuvai turėjo bendrą kombinaciją, jie turėjo pirmenybę, nes darbe norima palyginti ir tų pačių moduliacijų, bet skirtingų RS pereinamuosius signalus.

10 lentelė. RFM22B ir RFM69HW eksperimente naudotos moduliacijos ir jų parametrai

Siųstuvo modelis	Moduliacija	Sparta, kbps	Dažnio nuokrypis, kHz
RFM22B	FSK	2	5
		38.4	19.6
		125	125
	GFSK	2	5
		19.2	9.6
		125	125
RFM69HW	FSK	2	5
		9.6	19.2
		125	125
	GFSK	2	5
		19.2	9.6
		125	125

Kadangi naudota biblioteka turėjo tik LoRa palaikymą RFM95W RS, siuntimui buvo panaudota LoRa moduliacija su skirtingais sklaidos ir pločio parametrais. Taip pat šiame tyrime bus pažvelgta ir į LoRa moduliacijos pereinamąjį signalą ir skirtumus tarp FSK/GFSK moduliacijos.

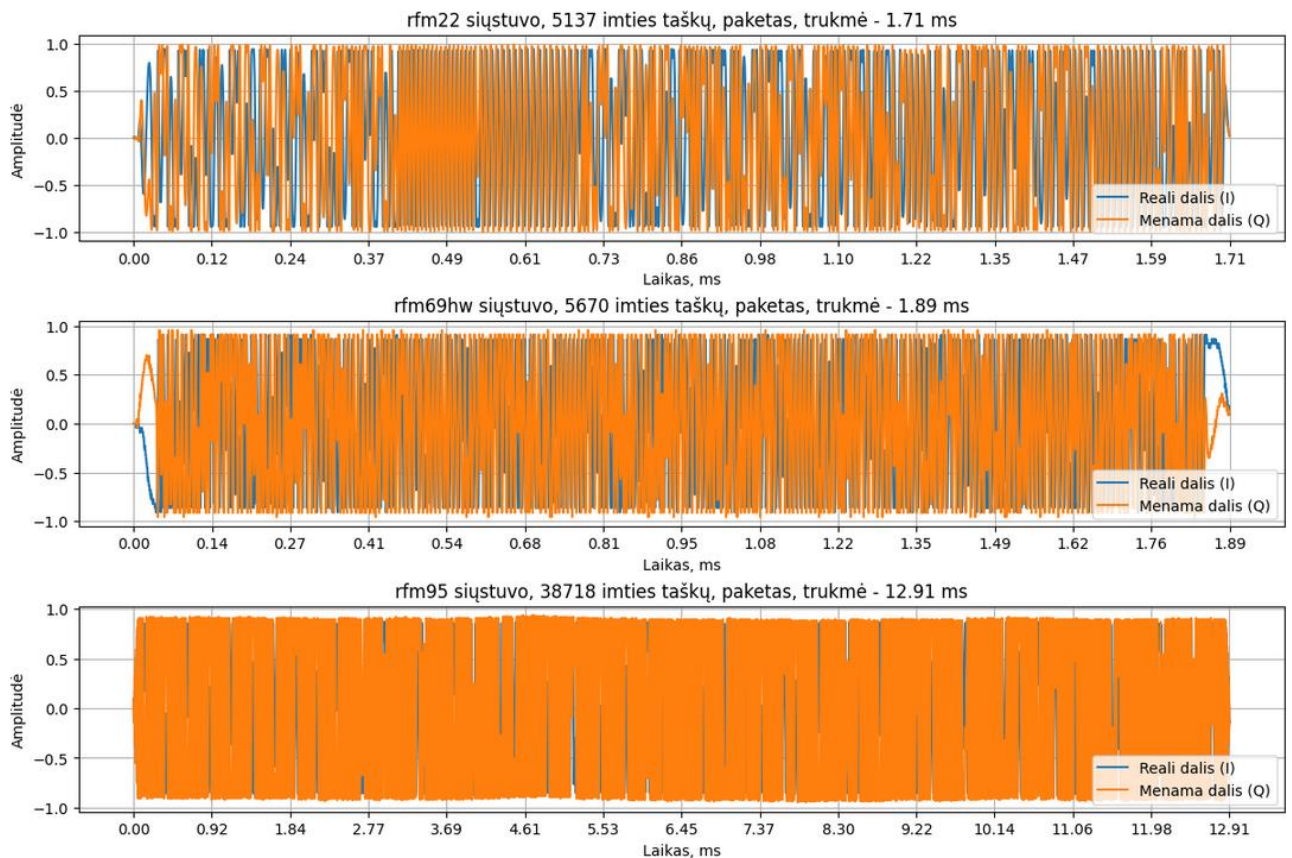
11 lentelė. RFM95HW eksperimente naudotos moduliacijos ir jų parametrai

Siųstuvo modelis	Moduliacija	Dažnių juostos plotis, kHz	Sklaida
RFM95HW	LoRa	31,25	SF 9
		125	SF 11
		500	SF 7

3.1. Analizuojamų duomenų apžvalga

Analizuojami patys paketai ir jų laikinė sritis. Visi signalų duomenys susideda iš realios I (angl. *In phase*) ir menamos dalių Q (angl. *Quadrature*), literatūroje vadinamais I/Q. Taip pat, dėl skirtingų išsiuntimo galios parametrų, visi signalai yra normalizuoti [-1; 1] režiuose. Paketų aprašymuose ID nurodo, kelintas paketas iš įrašytų yra naudojamas.

Taigi, pateikiama RS išsiųstų paketų laiko sritis, kur RFM22B ir RFM69HW naudojama FSK moduliacija, sparta 125 kbps, o dažnio nuokrypis 125 kHz, o RFM95W naudojama LoRa moduliacija, kur vieno paketo užimama dažnių juosta yra 500 kHz, o sklaida SF7, tai yra didžiausią spartą turinti kombinacija naudojant šį RS.

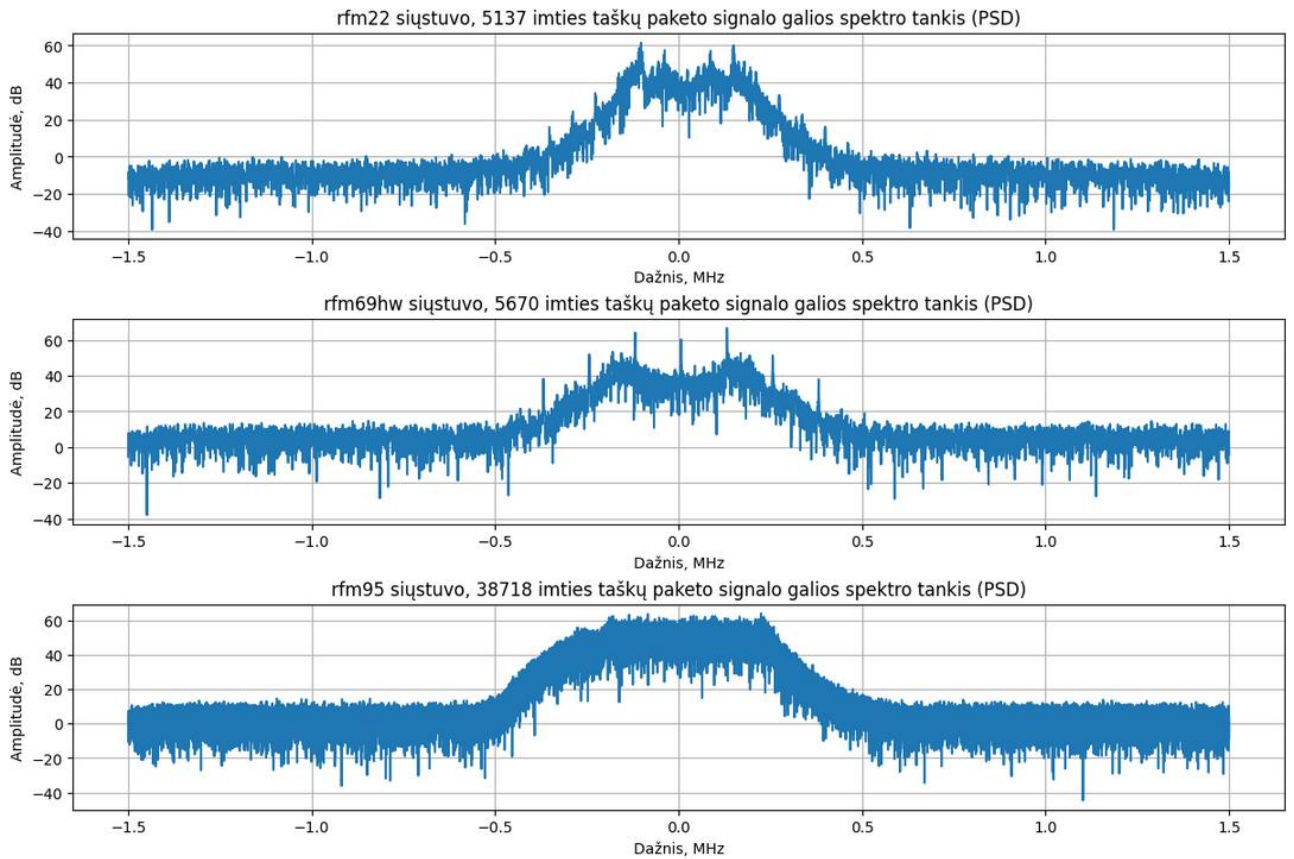


15 pav. Skirtingų radijo siųstuvų paketų pavyzdžiai laiko srityje

Lyginant RFM95W su RFM22B ir RFM69HCW laiko srityje (žr. 15 pav.) matyti, jog net ir siunčiant didžiausio pločio LoRa moduliaciją, imties taškų santykis yra apie 7 kartus didesnis lyginant su FSK ir GFSK moduliacijomis. Siunčiant mažesnio pralaidumo juostomis, šis santykis didėja, o tai yra vienas iš RS klasifikavimo minusų, bet tuo pačiu ir pliusų, nes, naudojant pačio paketo informaciją ir turint didesnę kiekį duomenų, galima kurti sudėtingesnius algoritmus, kurie galbūt tiksliau suklasifikuotų pačius RS pagal iš pačio duomenų paketo ištrauktus požymius, t.y. požymių pasirinkimas tampa didesnis, tačiau, norint apdoroti tokį kiekį duomenų, reikia turėti galingą duomenų rinkimo ir klasifikavimo sistemą, kuri galėtų veikti realiu laiku, o tai kelia sistemos kainą ir sunkina jos realizavimą. Tokios sistemos dažniausiai realizuojamos naudojamos FPGA ar iš anksto suprojektavus ASIC sistemas, nes, naudojant šias technologijas, galima sulygiagretinti skaičiavimus, o skaičiavimų trukmė yra numatoma iš anksto.

Taip pat pateikiamas analizuojamų RS, vieno atsitiktinai pasirinkto, paketo apskaičiuotas signalo galios spektro tankis PSD (angl. *Power spectral density*) (žr. 16 pav.) pagal pateiktus laiko srities I/Q duomenis (žr. 15 pav.). Iš PSD jau matyti, jog visų RS spektrai yra skirtingi. Netgi laiko srityje pateikiami paketai yra šiek tiek ilgesni ar trumpesni priklausomai nuo RS.

Kadangi diskretizavimo dažnis yra 3 MHz, o pagal Naikvisto teoremą, naudinga informacija yra tik iki pusės dažnio, t.y. 1.5 MHz, tačiau dėl kompleksinių duomenų (vienas imties taškas turi du duomenų vienetus), neigiami dažniai taip pat gali nešti informaciją, o tai leidžia daryti išvadą, kad visas 3 MHz ruožas yra naudingi duomenys.

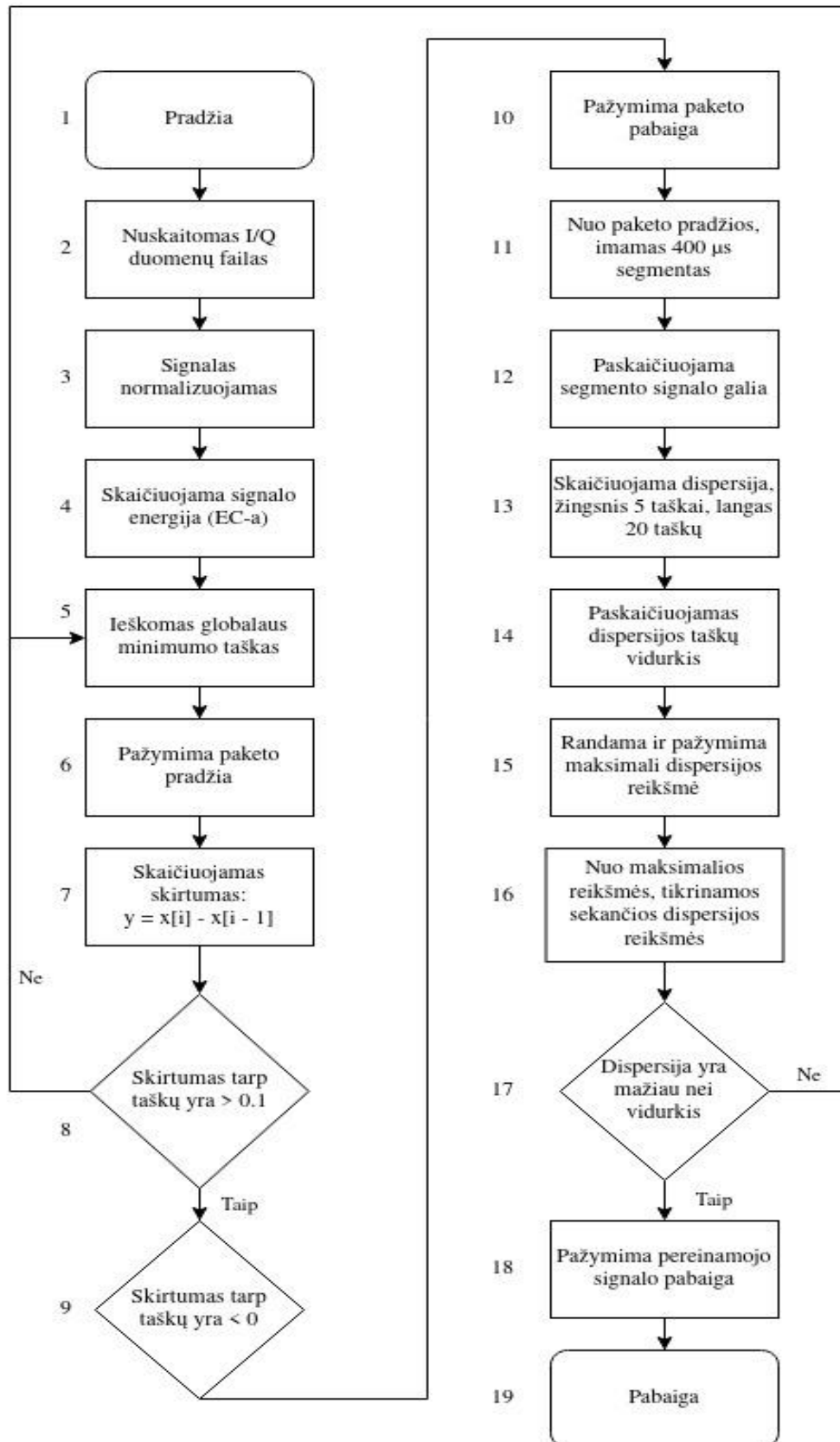


16 pav. Skirtingų radijo siųstuvų, atsitiktinai parinktų paketų, PSD spektras

Taip pat, pateikiamos ir RS spektrogramos (žr. 1 priedą), kuriuose vaizduojama keletas išsiųstų ir įrašytų paketų. Spektrogramose matomas centrinio dažnio poslinkis bei veidrodinė interferencija, kuri atsiranda dėl heterodino panaudojimo SDR imtuve. Įprastai, ši interferencija turėtų būti nufiltruojama juostos pralaidumo filtrais.

3.2. Pereinamųjų signalų analizė

Pateikiamas supaprastintas paketo pradžios ir galo detektavimo algoritmo aprašymas. Pagal numeraciją, detalizuojamas kiekvienas punktas



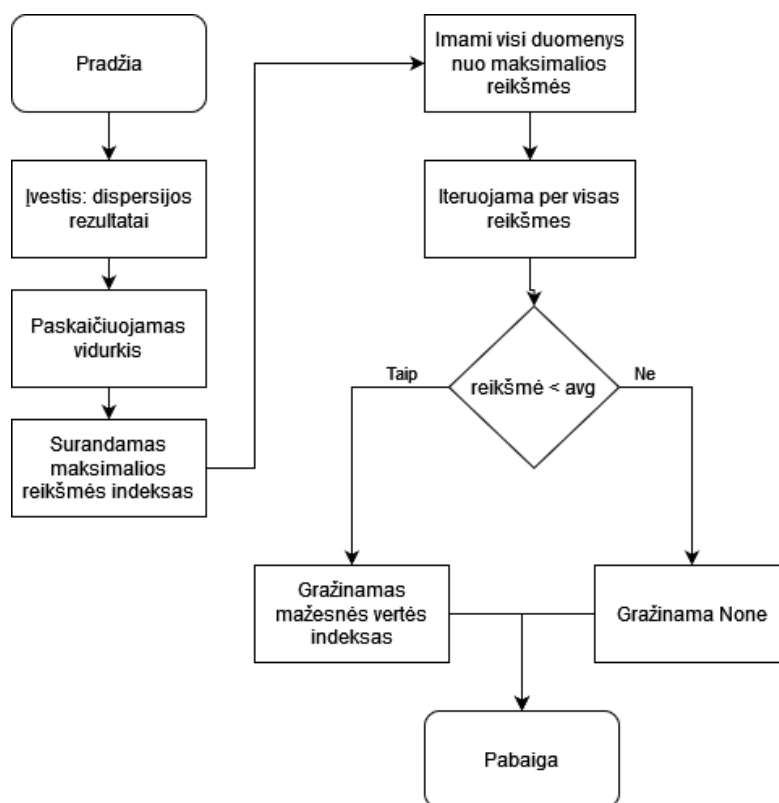
17 pav. Pereinamojo signalo ištraukimo iš laiko srities duomenų algoritmas

1. Pradžia (naudojamas *Python* programavimo kalbos skriptas);
2. Nuskaitomas pateiktas duomenų failas, duomenys yra I/Q formato, duomenų diskretizavimo dažnis yra 3 MHz;
3. Prieš atliekant signalo energijos skaičiavimus, visas duomenų failas yra normalizuojamas – šiuo metu, tai pradžios nustatymo algoritmo EC- α , reikalavimas;
4. Pagal 7, 8 ir 9 formules paskaičiuojama signalo energija ir energijos trajektorija;
5. Ieškomas globalus minimumas, kuris ir bus pereinamojo signalo pradžios taškas – signalas, iki šios vietos, yra traktuojamas kaip triukšmas ir yra iškerpamas;
6. Išsaugomas minimumo indeksas, nes tai – pereinamojo signalo pradžios taškas (pagal EC- α algoritmą) ir jis bus naudojamas pereinamajai daliai ištraukti;
7. Skaičiuojamas skirtumas tarp signalo gretimų energijos taškų;
8. Sekanti algoritmo dalis aktyvuojasi, jeigu skirtumas tarp gretimų energijos taškų yra daugiau nei 0.1, nes gali būti, jog skaičiuojant nuo minimum taško, skirtumas bus neigiamas, tad yra naudojamas slenkstis, kurį peržengus, pereinama į sekantį tikrinimą;
9. Kai energijos kriterijaus reikšmės pasiekia maksimumą (baigiasi paketas - pradeda mažėti energija), skirtumo reikšmės pradeda kristi žemiau 0, nes vėl ieškoma sekančio globalaus minimumo. Ši vieta, kur yra kertamas 0, yra paketo pabaiga;
10. Ta vieta, kur randamas skirtumas yra mažiau už 0, yra pažymima kaip paketo pabaiga;
11. Sekantis žingsnis yra surasti pereinamojo signalo galą. Pirmiausiai, kad nereikėtų skaičiuoti viso paketo statistinių verčių, nuo paketo pradžios yra imamas 400 mikro sekundžių segmentas, tai turėtų būti bent 5 kartus daugiau nei bet koks šiame darba analizuojamų paketų pereinamasis signalas. Šioje vietoje segmentas negali būti per ilgas, nes tuomet sumažėja dispersijos vidurkis, o tai gali daryti įtaką pereinamojo signalo pabaigos nustatymui;
12. Segmentas yra kompleksinių verčių, tad paskaičiuojama jo galia;
13. Pereinamasis signalas turi daug dedamųjų ir jo dispersija kiekviename taške nuo 0 iki maksimalios amplitudės yra kintanti, tačiau kai pasiekiamas maksimali amplitudė, dispersija susinormalizuoja.
14. Šioje vietoje paskaičiuojama segmento vidutinė dispersija;
15. Dėl pereinamojo signalo, atsiranda dispersijos pikas, kuris, dažniausiai, turėtų būti pereinamojo signalo vidurys;
16. Ieškant nuo maksimalios vertės indekso, pereinamosios signalo dalies galas pažymimas tas, kur dispersija pasiekia vidutinę dispersijos reikšmę;
17. Išsaugoma pereinamojo signalo pabaiga;
18. Pabaiga: apdorota energija yra paslenkama per viso paketo ilgį ir grįžtame į 5 - tą žingsnį, kuriame ir vėl ieškome globalaus minimumo, t.y. naujo paketo pradžios.

Algoritmo veikimo vizualizacijos pateikiamos 19 pav., 20 pav. ir 21 pav.. Juose vaizduojama pagal 13-ą žingsnį aprašyta dispersija, kur raudona linija žymi pereinamojo signalo pabaigą. Matyti, jog pasinaudojant dispersijos skaičiavimu, galima išgauti gana tikslią pabaigą – ištrauktas pereinamasis signalas ir jo trukmė yra taip pat pateikiami. Atitinkamai, pateikiama ir signalo amplitudė, kuri paskaičiuojama pagal (20-ą formulę. Pereinamojo signalo pabaigos imties numeris išskaičiuojamas pasinaudojant dispersijos formulę, kur įvestis yra paskaičiuota amplitudė:

$$dispersija[X] = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 \quad (19)$$

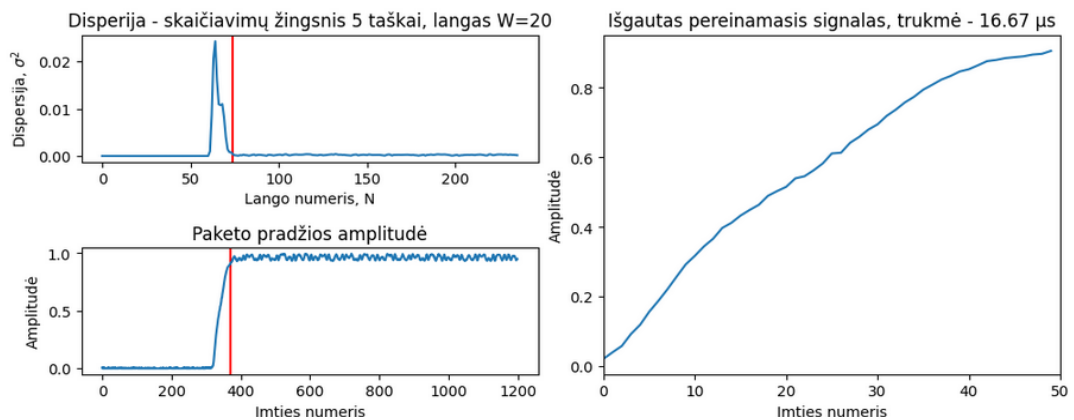
čia \bar{x} – duomenų rinkinio vidurkis, o N – duomenų imties kiekis. Dispersija skaičiuojama kas 5 imties taškus, o naudojamas langas yra 20 imties taškų. Dėl pereinamojo signalo sąvybių – greitai kintanti amplitudė, kuri nusistovi tik pasiekus maksimalią paketo amplitudę, tad naudojant dispersijos metodą galima detektuoti, kada signalo amplitudė nustoja kisti. Pereinamojo signalo pabaigos nustatymas pateikiamas 18 pav.



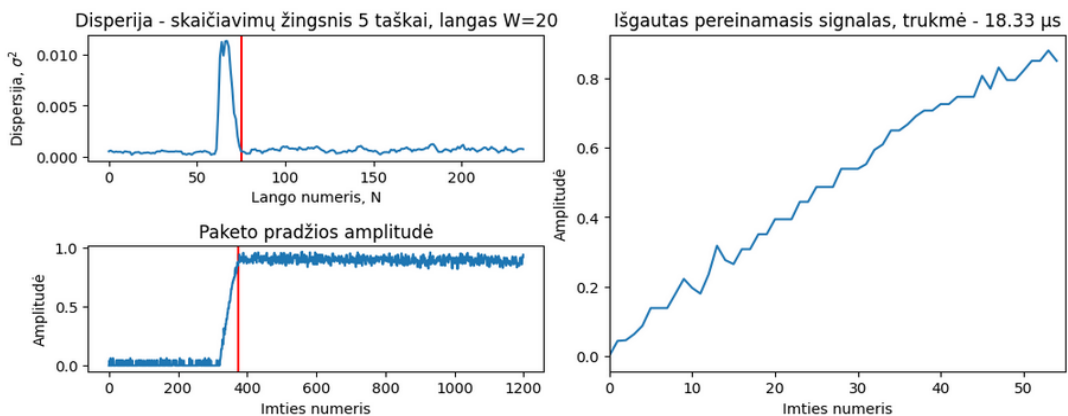
18 pav. Pereinamojo signalo pabaigos nustatymas, kai įvestis yra dispersijos reikšmių masyvas

Analizuojant aprašyto algoritmo veikimą, pirmiausiai analizuojamas 19 pav., kuriame matyti dispersijos ir amplitudės reikšmės. Dispersijos ir amplitudės reikšmių grafikai x ašyse yra suvienodinti – lango numeris koreliuoja su imties numeriu. Tarpusavyje lyginant šiuos du grafikus, amplitudės grafike matomas staigus jos kilimas nuo 300 iki 400 imties taško, o toliau stebima stabili amplitudė. Toks amplitudės pokytis atsispindi dispersijos grafike – amplitudei kintant, dispersija didėja.

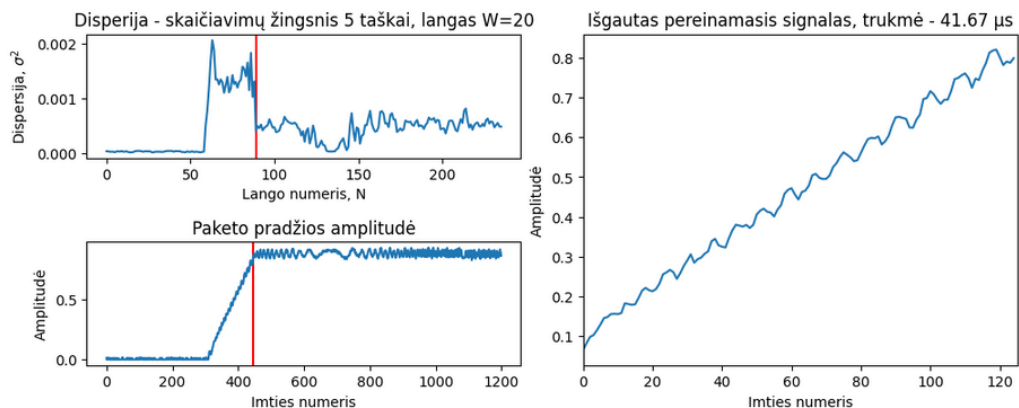
Lyginant RFM22B ir RFM69HW FSK moduliacijos paketus, matyti, jog jų amplitudės ir dispersijos grafikai yra panašūs, tačiau RFM95HW RS dispersijos grafikas yra labiau kintantis, tačiau nepaisant to, nustatomas gana tiksli pereinamojo signalo pabaiga. Taigi, iš RFM95HW RS rezultatų galime daryti išvadą, jog kai kuriais atvejais, dispersijos metodas gali netikti.



19 pav. RFM22B radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija



20 pav. RFM69HW radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija

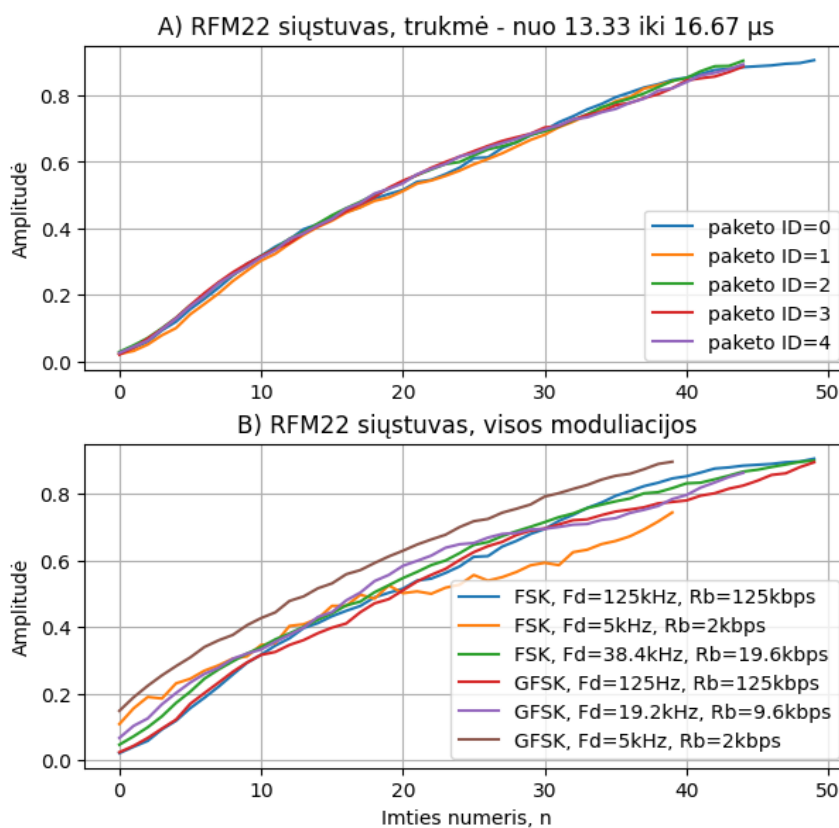


21 pav. RFM95HW radijo siųstuvo pereinamojo signalo ištraukimo skaičiavimo vizualizacija

Išnagrinėjus keletą skirtingų RS duomenų paketų I/Q ir PSD formose, analizė perkeliama į pereinamųjų signalų analizavimą. Šiame skyriuje apžvelgiami to paties siųstuvo pereinamieji signalai, atliekamas to pačio siųstuvo pereinamųjų signalų palyginimas esant skirtingai moduliacijai bei palyginami skirtingų RS pereinamieji signalai. Lyginant to paties siųstuvo pereinamuosius signalus, RFM22B, RFM69HW ir RFM95W naudoja ankstesniame skyriuje paminėtas moduliacijas ir RS konfigūracijas. Paveiksluose pateikiama realios ir menamos reikšmių absoliutinė reikšmė, nurodanti, kaip kintant laikui, kinta signalo galia, kuri apskaičiuojama pagal formulę:

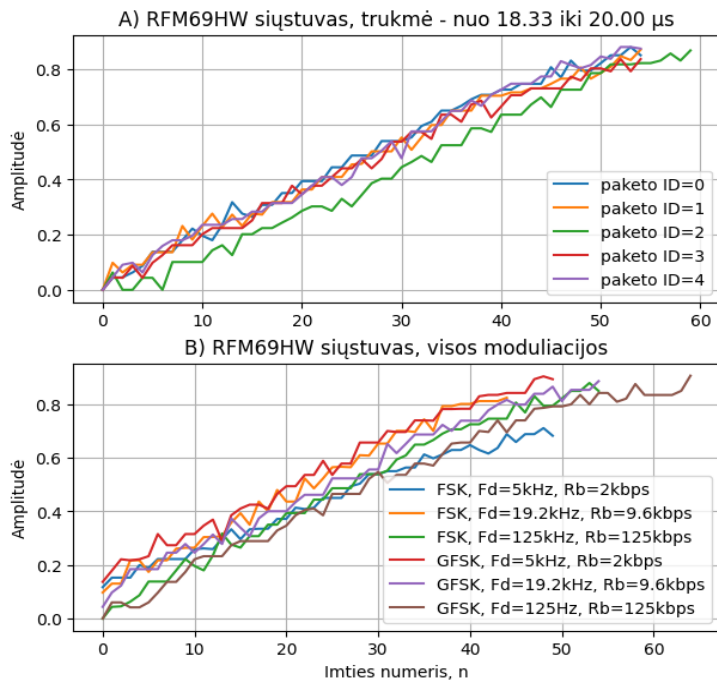
$$|z| = \sqrt{Re(z)^2 + Im(z)^2} = \sqrt{I^2 + Q^2} \quad (20)$$

Pateikiami trys RFM22B pereinamieji signalai laiko srityje (žr. 22 pav.). Šalia pateikiamos ir visos šio siųstuvo moduliacijos su skirtingomis dažnio poslinkio ir spartos reikšmėmis. Analizuojant tos pačios moduliacijos, bet skirtingų paketų signalus, matyti, jog amplitudė tam tikrose laiko momentuose skiriasi, tačiau pati forma yra gana panaši, tačiau, analizuojant visų moduliacijų signalus, matomas gana aiškus trukmių skirtumas tarp žemiausios ir aukščiausios spartos signalų, pereinamojo signalo pradžios, nustatymo algoritmo.



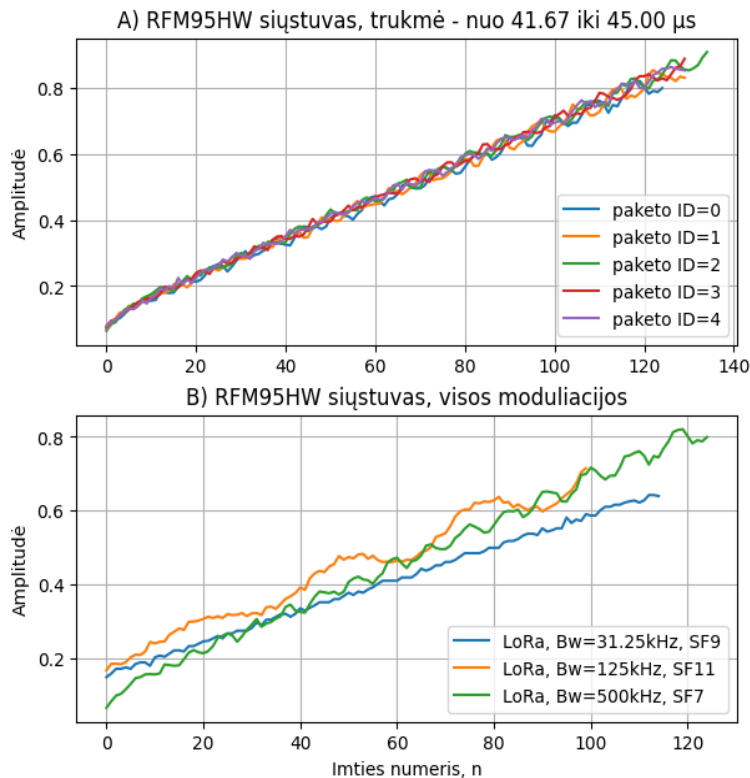
22 pav. RFM22B radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai

Tolimesniam palyginimui pateikiami 5 RFM69HW pereinamieji signalai (žr. 23 pav.). Šiuose signaluose matyti, jog pereinamieji signalai skiriasi nuo RFM22B siųstuvo signalų - turi didesnę duomenų dispersiją. Taip pat matomas ir didesnis skirtumas tarp to paties siųstuvo, bet skirtingų duomenų paketų. Stebima ta pati trumpesnių signalų tendencija kaip ir su RFM22B RS – abiejų RS pereinamųjų signalų trukmė yra apie 50 μ s.



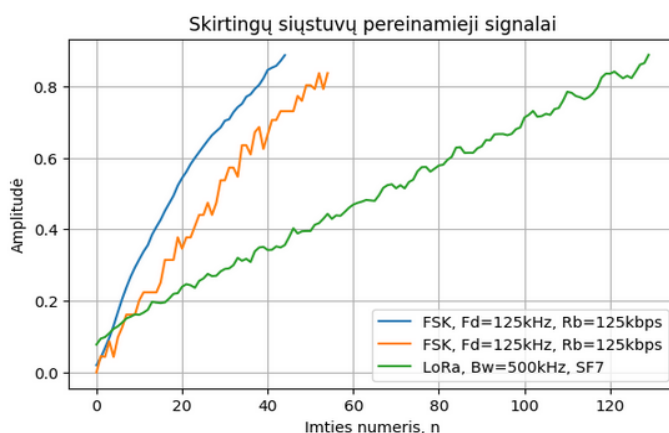
23 pav. RFM69HW radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai

Paskutinis analizuojamas RS yra RFM95W, kurio analizavimai buvo taip pat panaudoti 5 pereinamieji signalai (žr. 24 pav.). Gana žymus požymis yra tai, jog, lyginant su kitais siųstuvais, šio RS pereinamasis signalas yra bent du kartus ilgesnis. Šio siųstuvo pereinamieji signalai tarp paketų yra šiek tiek panašūs, tačiau atskiras signalas turi nemažą signalo amplitudės dispersiją.



24 pav. RFM95HW radijo siųstuvo paketų laiko srities pereinamieji signalai. A) Tos pačios moduliacijos pereinamieji signalai, B) Skirtingų moduliacijų pereinamieji signalai

Palyginus kiekvieno siųstuvo pereinamuosius signalus, pateikiamas ir grafikas, kuriame lyginamas kiekvieno atskiro RS vienas atsitiktinis signalas su skirtingais RS (žr. 25 pav.). Matyti, jog net ir laiko srityje pereinamieji signalai turi skirtumų, tačiau, didėjant duomenų ir RS kiekiui, šie požymiai gali labai greitai susivienodinti [15].



25 pav. RFM22B, RFM69HW ir RFM95HW radijo siųstuvų pereinamieji signalai viename grafike

Taip pat, buvo paskaičiuota visų RS pereinamųjų signalų vidutiniai ilgiai, kur RFM22B ir RFM69HW rezultatai pateikiami 12 lentelėje, o RFM95HW pateikiami 13 lentelėje. Pagal lentelių duomenis, skirtingų RS ir pereinamųjų signalų vidutinės trukmės yra skirtingos – lyginant RFM22B ir RFM95HW, RFM69HW signalai ilgesni nuo 3 iki 11,2 μ s, o lyginant RFM69HW ir RFM95HW, RFM95HW signalai ilgesni nuo 21,8 iki 26,6 μ s.

12 lentelė. RFM22B ir RFM69HW pereinamųjų signalų vidutinės trukmės

Siųstuvo modelis	Moduliacija	Dažnio nuokrypis, kHz	Vidutinė trukmė, μ s
RFM22B	FSK	5	13,7
		19.6	14,9
		125	14,9
	GFSK	5	13,7
		9.6	14,3
		125	15,2
RFM69HW	FSK	5	17,3
		19.2	17,4
		125	24,9
	GFSK	5	16,7
		9.6	17,9
		125	20,7

Prieš tęsiant Hilberto-Huango analizę, 28 pav. pateikiamas HHT ir STFT metodų palyginimas. Naudojama GFSK moduliacija, o dažnio poslinkis yra 19,2 kHz. Šio signalo diskretizavimo dažnis yra 3 MHz, o STFT algoritmui parinkta 512 segmentų, o tai reiškia, jog algoritmo raiška yra 5859,375 Hz. Toks segmentuotas vaizdas ir yra matomas STFT dalyje, o analizuojant HHT spektrą, gana aiškiai matomas energijos kiekis tam tikru laiku, tam tikrame dažnyje.

13 lentelė. RFM95HW pereinamųjų signalų vidutinės trukmės

Siųstuvo modelis	Moduliacija	Dažnių juostos plotis, kHz	Vidutinė trukmė, μ s
RFM95HW	LoRa	31,25	38,6
		125	38,5
		500	43,2

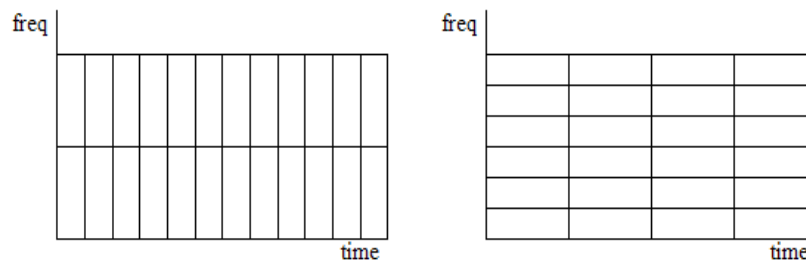
Atlikus apžvalginę RS laiko ir dažnių srities analizę, galima daryti išvadą, jog šiame darbe analizuojamų RS pereinamieji signalai yra skirtingi ir yra tinkami neuroninio tinklo apmokymui.

3.3. Pereinamųjų signalų laiko – dažnio – energijos pasiskirstymo analizė

Duomenų paketo pereinamasis signalas nėra stacionarus, tad ši dalis negali būti analizuojama paprastąja Furjė transformacija – turi būti naudojamas metodas, kuris galėtų išanalizuoti netiesinius ir nestacionarius duomenis. Vienas iš galimų variantų yra STFT panaudojimas, kai N_ω nurodo lango $w[n]$ ilgį, N nurodo diskretinių dažnių kiekį, o dažniai, kurie bus naudojami STFT pateikiami kaip $\omega_k = 2\pi k/N$. Kai naudojamas apibrėžto dydžio langas nuo 0 iki $N_\omega - 1$, gaunama formulė:

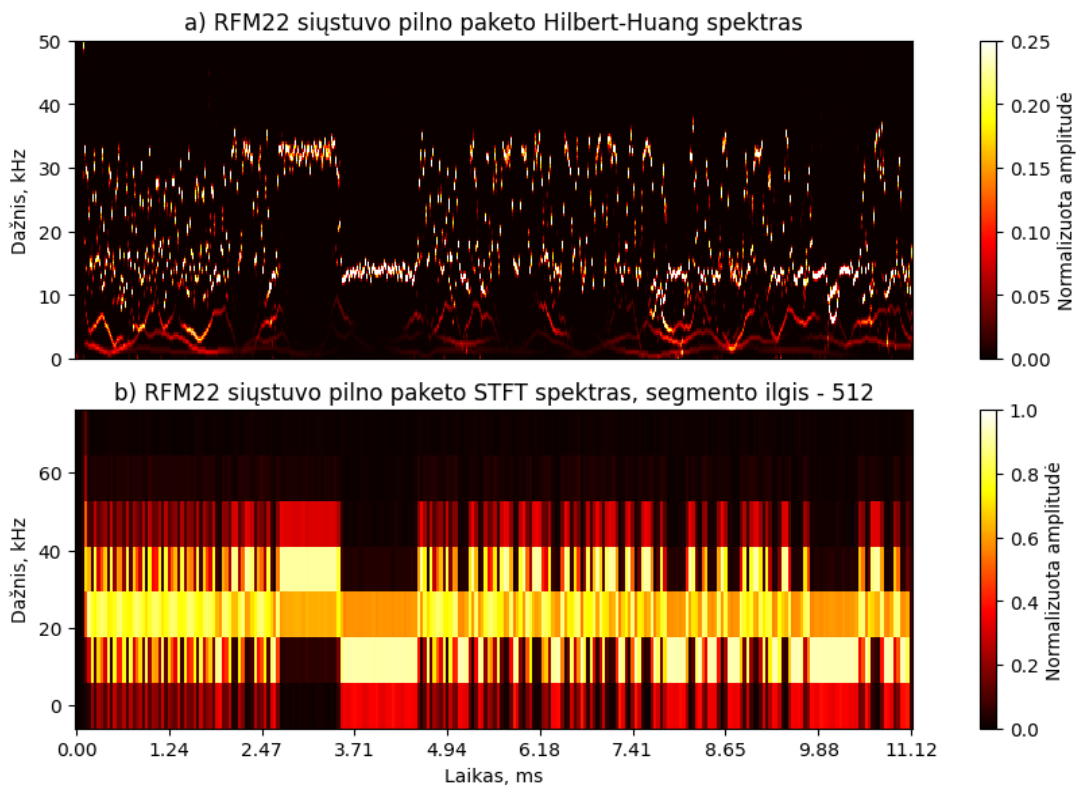
$$X[n, k] = \sum_{m=n-(N_\omega-1)}^n w[n-m]x[m]e^{-\frac{j2\pi mk}{N}} \quad (21)$$

STFT yra gana greitas metodas, kurio algoritmo sudėtingumas gali siekti ir $O(N \log_2 N)$ (metodo greitaveika aprašoma pagal didžiosios O notaciją), tačiau jis turi vieną didelį minusą, dėl ko negali būti efektyviai panaudojamas pereinamojo signalo analizėje - metodas turi fiksuotą rezoliuciją, t.y. vienu metu gali būti tik gera dažninė arba tik laikinė rezoliucija. Platus langas suteikia gerą dažninę rezoliuciją, tačiau blogą laikinę ir atvirkščiai (žr. 26 pav.). Ši ypatybė yra susijusi su Heizenbergo principu.



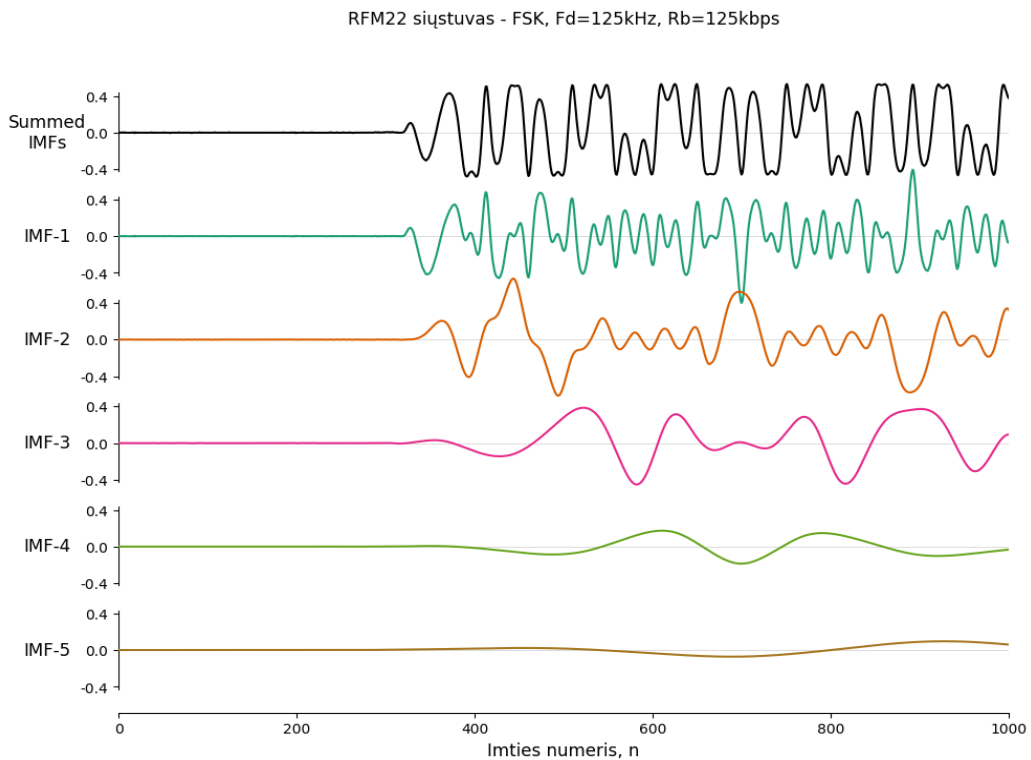
26 pav. STFT raiškos palyginimas. Kairėje – geresnė laiko raiška, dešinėje – geresnė dažninė raiška

Būtent dėl šios priežasties buvo ieškoma naujų analizės metodų: diskretinių vilnelių transformacijos ar Hilberto-Huango transformacijos panaudojimas. Šiam darbui pasirinktas metodas yra Hilberto-Huango transformacija. Tai empirinis ir adaptyvus metodas, skirtas būtent nestacionarių duomenų analizei, todėl nereikia žinoti jokios išankstinės informacijos apie analizuojamus duomenis. Metodas suteikia gana tikslų amplitudės pasiskirstymą kintant laikui ir dažniui. Naudojant šį metodą, realizuojamas pereinamojo signalo klasifikavimas.



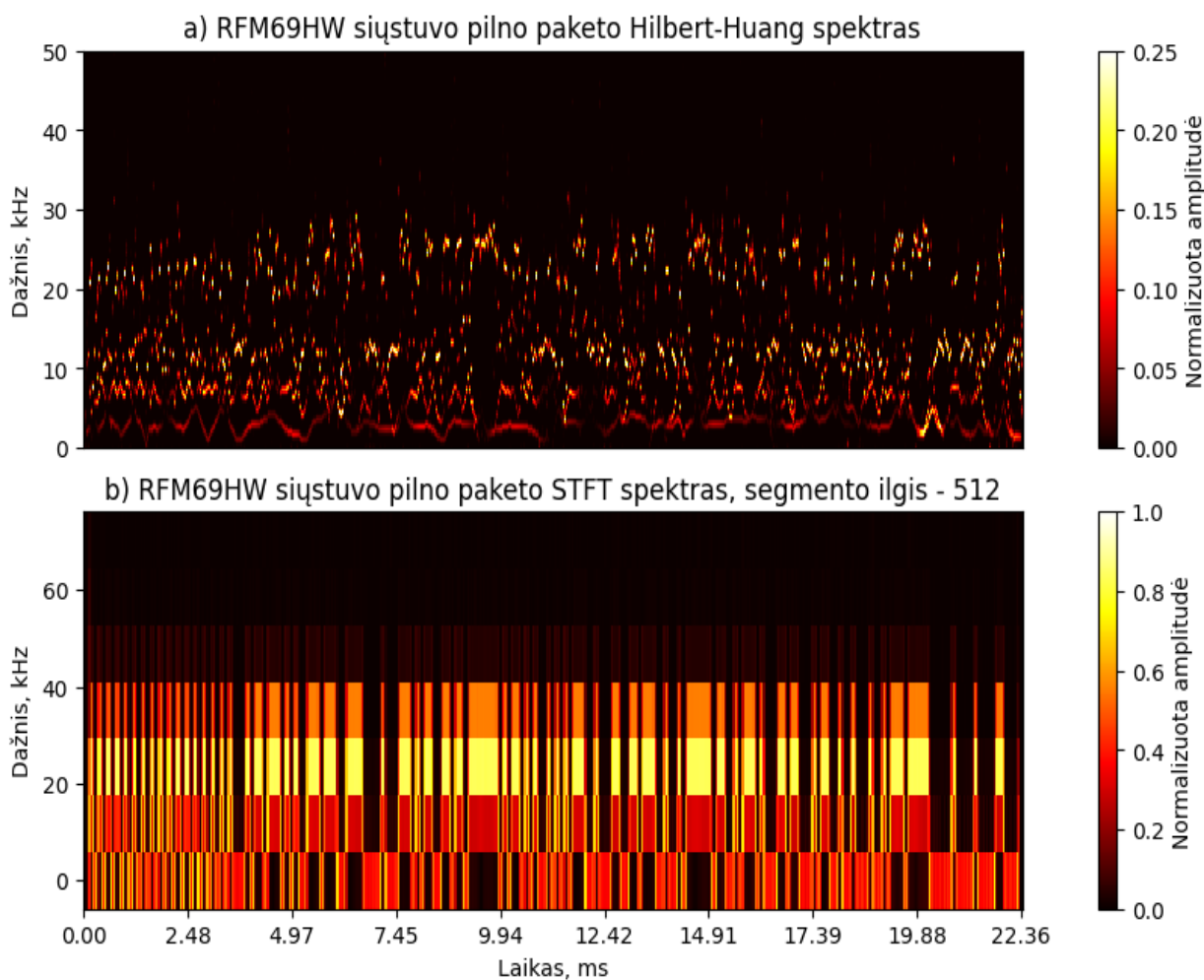
28 pav. RFM22 siųstuvo Hilberto-Huang ir STFT transformacijų palyginimas

Išskaičiuotos IMF komponentės pateikiamos 27 pav. Iš atliktos analizės matyti, jog signalas susideda iš 5-ių skirtingų dedamųjų signalų, kur IMF-1 yra aukščiausio dažnio, o IMF-5 – žemiausio. Norint detaliau išnagrinėti Hilberto spektro pateiktą informaciją, detektuojama ir ištraukiama pereinamojo signalo dalis.



27 pav. RFM22B radijo siųstuvo duomenų paketo laiko srities signalų

Pateikiamas RFM69HW RS palyginimas (žr. 29 pav.), kuriame naudojama ta pati moduliacija, kaip ir RFM22 RS. Šiame pavyzdyje taip pat vizualiai matomas laiko–dažnio segmentavimas. STFT pavyzdyje didžiausia amplitudė stebima ties 20 – 30 kHz, tad nėra aišku, kurio dažnio amplitudė yra dominuojanti, o HHT spektre gana aiškiai matyti, jog maksimali amplitudė yra ties 20 kHz.

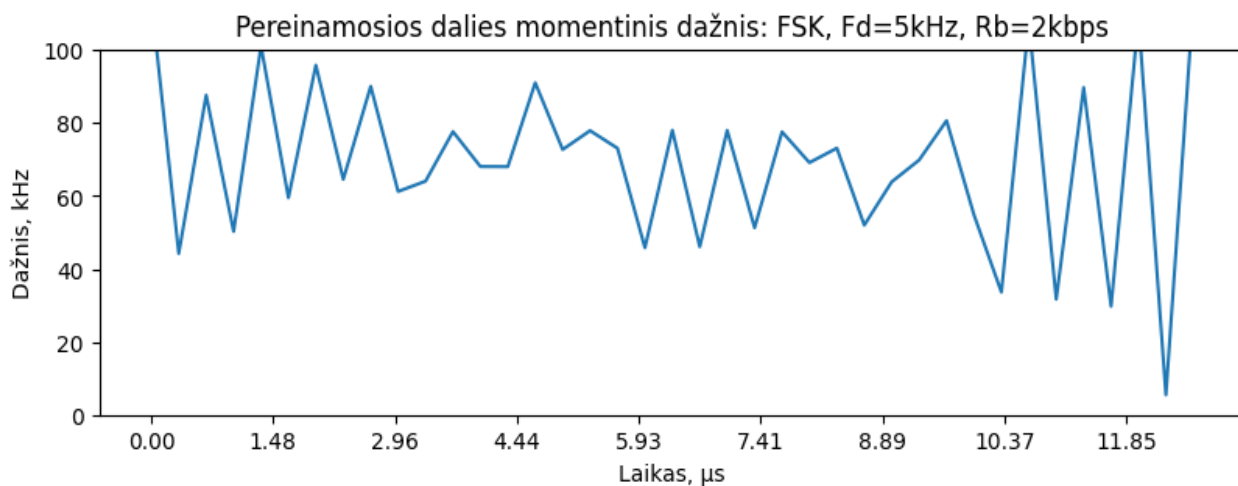


29 pav. RFM69HW siųstuvo Hilberto–Huango ir STFT transformacijų palyginimas

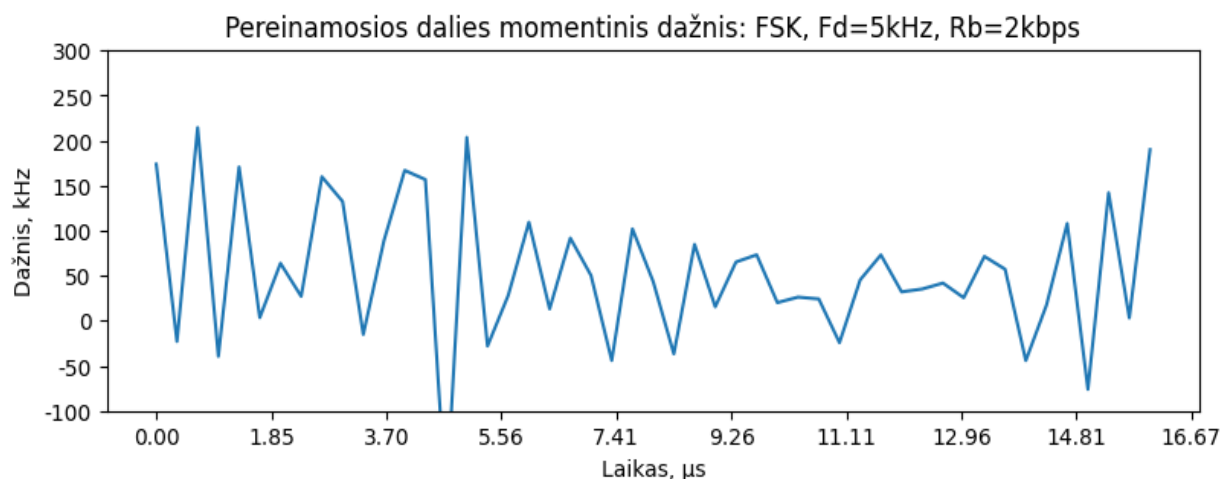
Palyginimui pateikiama ir paprastoji Hilberto transformacija, kurios įvestis yra realių reikšmių signalas. Pateikiamas RFM22B ir RFM69HW RS FSK moduliacijos momentinis dažnis (žr. 31 pav. ir 30 pav.), kuris yra gaunamas iš realaus signalo įvesties (Hilberto-Huango transformacijos IMF sumos atitikmens). Tuomet signalui pritaikoma Hilberto transformacija, kuri realų signalą paverčia į kompleksinį, pagal kurio reikšmes paskaičiuojamos momentinės fazės ir dažnio reikšmės.

Momentinis dažnis nusako, koks būtent tam tikru momentu buvo pačio signalo dažnis, tačiau jis nepateikia energijos kiekio, tad vien tik iš vizualios analizės nėra galimybės pasakyti, kuri dažnio komponentė yra dominuojanti.

Analizuojant momentinio dažnio grafikus, matyti, jog skiriasi ne vien tik duomenų ilgiai, bet ir momentinio dažnio vertės, nors tai ta pati moduliacija.



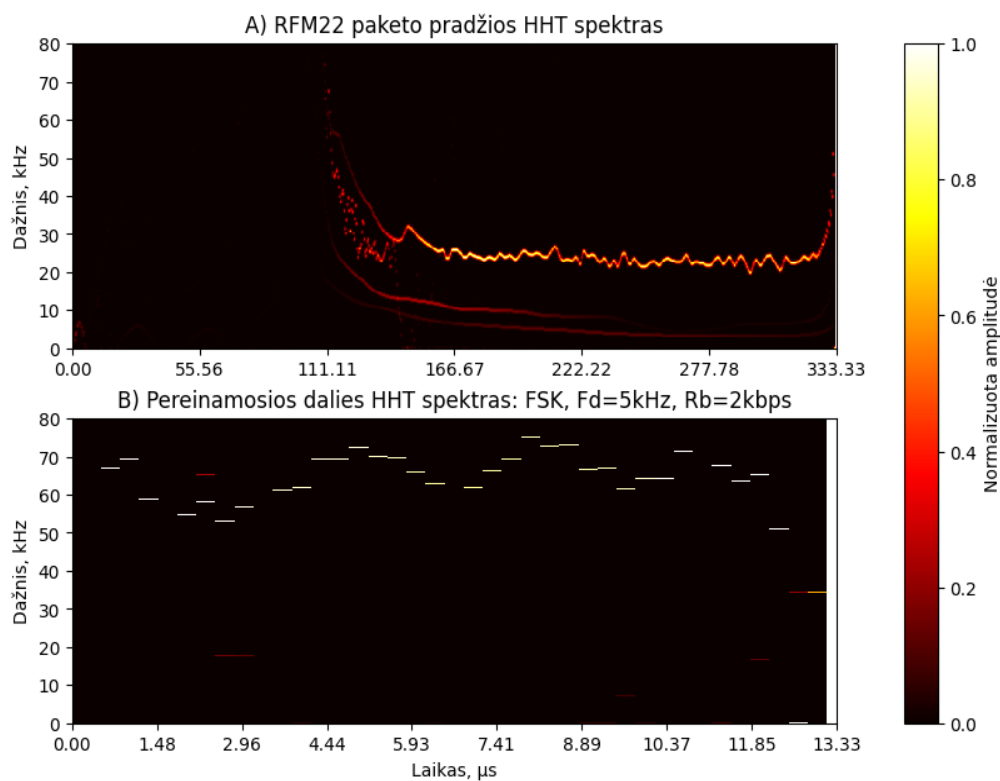
31 pav. RFM22B siųstuvo pereinamosios dalies momentinis dažnis esant FSK moduliacijai



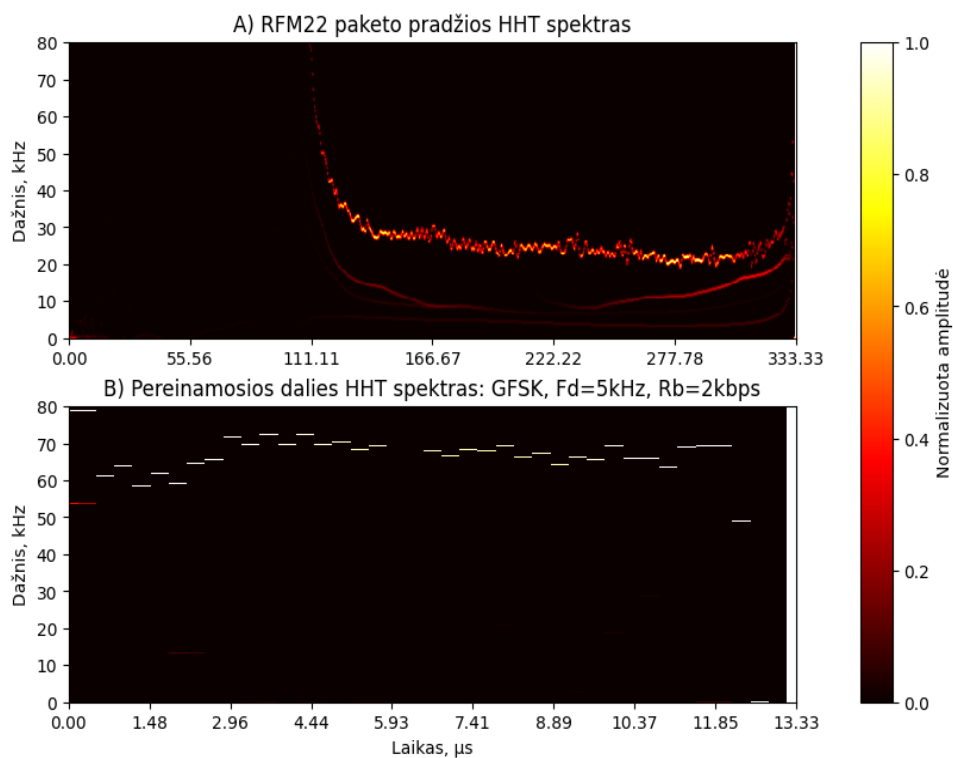
30 pav. RFM69HW siųstuvo pereinamosios dalies momentinis dažnis esant FSK moduliacijai

Išskaičiavus visas IMF komponentes (nurodžius IMF komponentių kiekio limitą), sekantis veiksmas yra kiekvienai IMF paskaičiuoti Hilberto transformaciją ir pateikti Hilberto–Huango spektrą (žr. 32 pav.). Taigi, pateikiama FSK moduliacijos rūšis, kai nuokrypio dažnis yra 5 kHz, taip pat, kitame paveiksle (žr. 33 pav.) pateikiama GFSK moduliacija su tokiu pačiu nuokrypiu.

Pateiktieji HHT spektrai indikuoja, jog RFM22B FSK ir GFSK moduliacijų spektrai yra labai panašūs – tai gali kelti problemų norint klasifikuoti skirtingas moduliacijas.

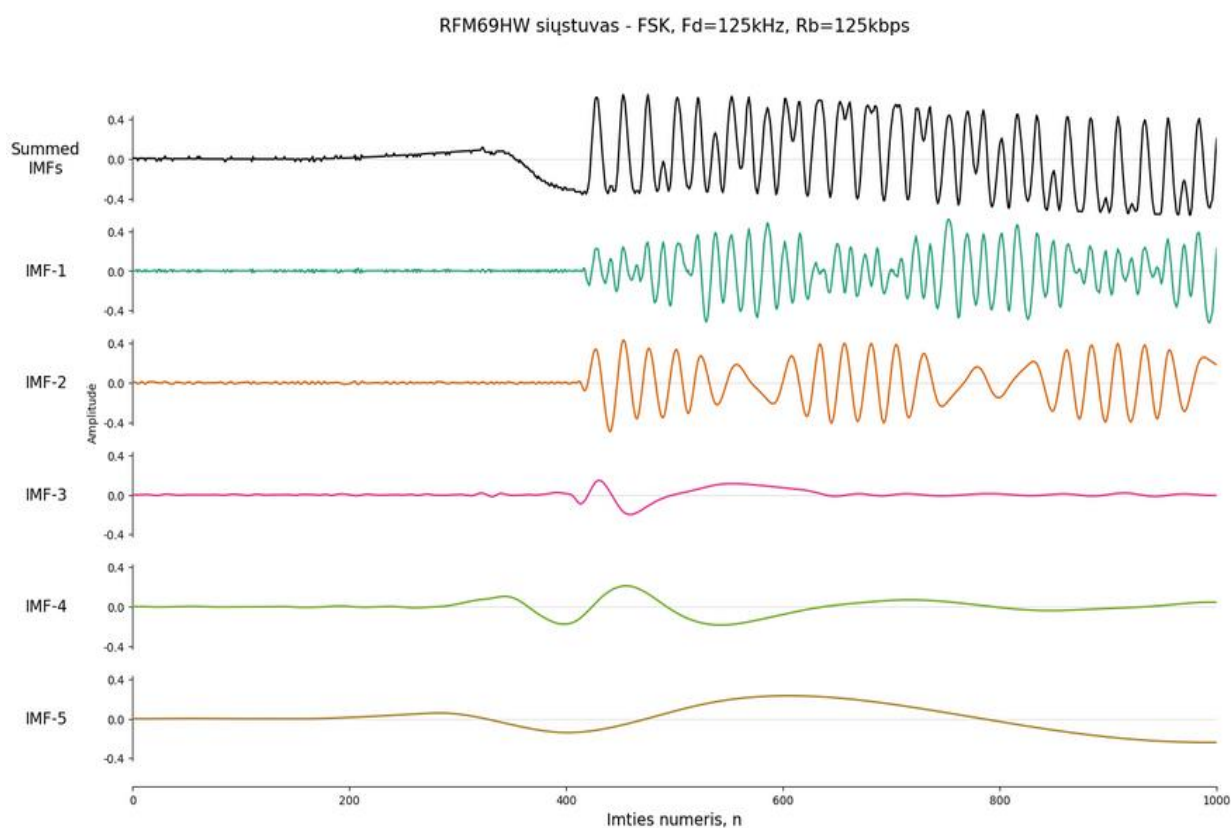


32 pav. RFM22B RS Hilberto-Huango FSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas



33 pav. RFM22B RS Hilberto-Huango GFSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas

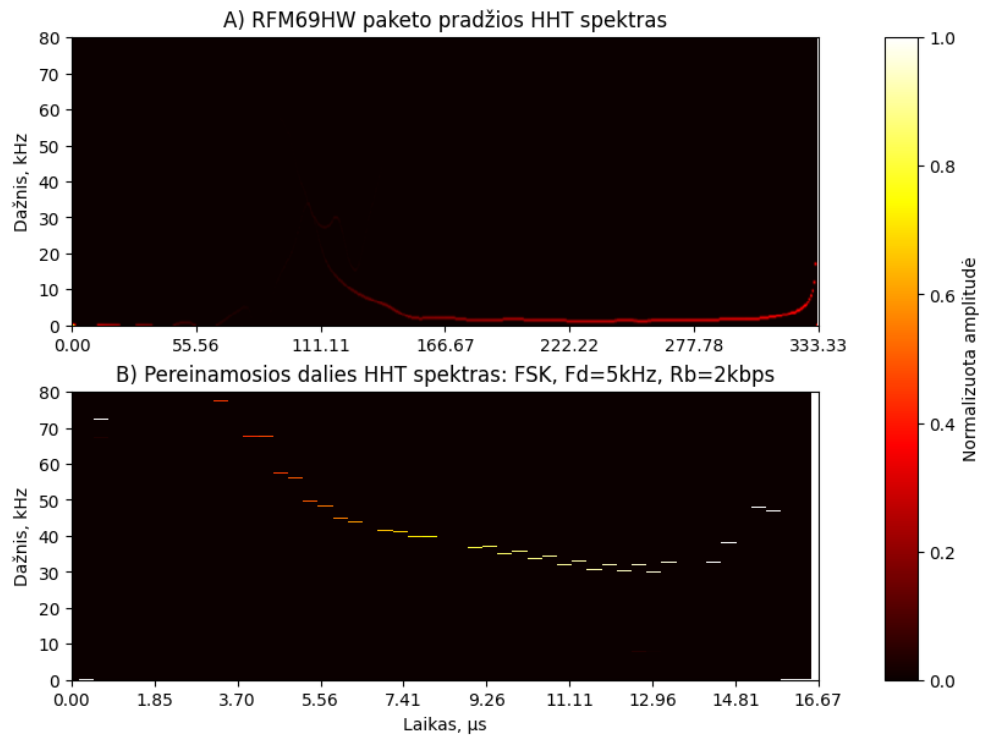
Analizuojant sekantį RS – RFM69HW, taip pat pateikiamas pilno duomenų signalo Hilberto spektras ir jo pereinamieji signalai (žr. 34 pav., 35 pav. ir 36 pav.). Analizuojant atskirus FSK moduliacijos IMF signalus, matyti, jog yra du pagrindiniai IMF (žr. 34 pav.), kurie ir sudaro didžiąją dalį energijos visame signale. Likę IMF-3, IMF-4 ir IMF-5 neturi moduluotos informacijos, jų didžiausia matoma amplitudė yra pačio signalo pradžioje – pereinamajame signale. Šiek tiek matoma ir žemų dažnių amplitudės pokyčių.



34 pav. RFM69HW radijo siųstuvo FSK moduliacijos duomenų paketo laiko srities signalų komponentės – IMF

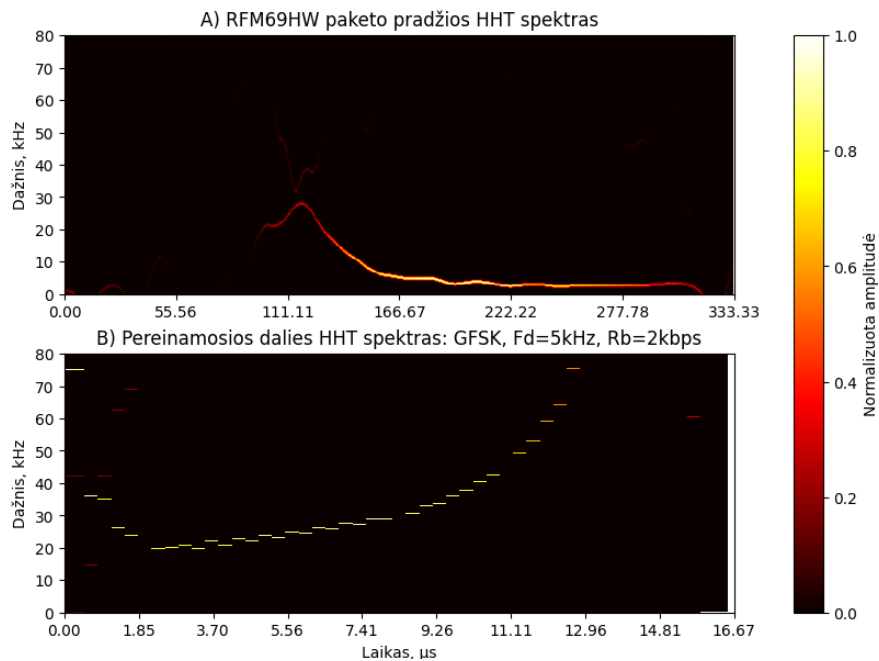
Analizuojant signalo pradžią (žr. 35 pav., A dalis), matyti, jog ties 119.05 μ s matomas vidutinio stiprumo energijos kiekis ties 20 kHz, tačiau nuo 142.86 iki 309.52 μ s energijos koncentracija pastoviai išsilaiko ties maždaug 5 kHz dažniu. Analizuojant patį pereinamąjį signalą (B dalis), matyti, jog signalo energijos trajektorija prasideda nuo 4.39 μ s ties 70 kHz su gana nežymiu energijos kiekiu, kuris didėja iki maksimumo dažniui artėjant link 30–40 kHz intervalo (nuo 7.02 iki 14.04 μ s), kur signalo energijos kiekis pasiekia maksimalią amplitudę.

Lyginant 32 pav. ir 33 pav. su 35 pav. ir 36 pav. paveikslais, matyti, jog spektrai yra skirtingi ir pačius RS vizualiai yra įmanoma atskirti. Lyginant RFM69HW FSK ir GFSK moduliacijas, taip pat matomas skirtumas. Reikia paminėti, jog šis siųstuvas naudoja tą pačią moduliaciją, su tais pačiais parametrais, kaip ir RFM22B RS, tačiau yra gaunamas kitoks Hilberto-Huango spektras – nors yra naudojamas tas pats 5 kHz dažnio nuokrypis FSK ir GFSK moduliacijose.



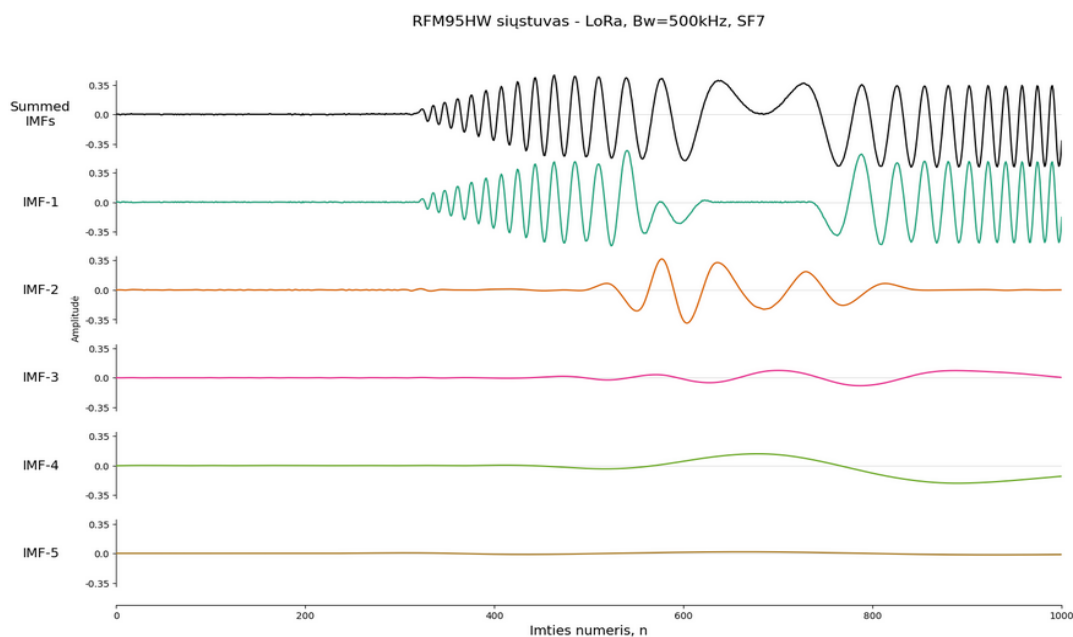
35 pav. RFM69HW RS Hilberto-Huango FSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas

Priešingai nei RFM22B, RFM69HW RS naudojant tą patį 5 kHz dažnio poslinkį, FSK ir GFSK moduliacijų HHT spektrai yra skirtingi.



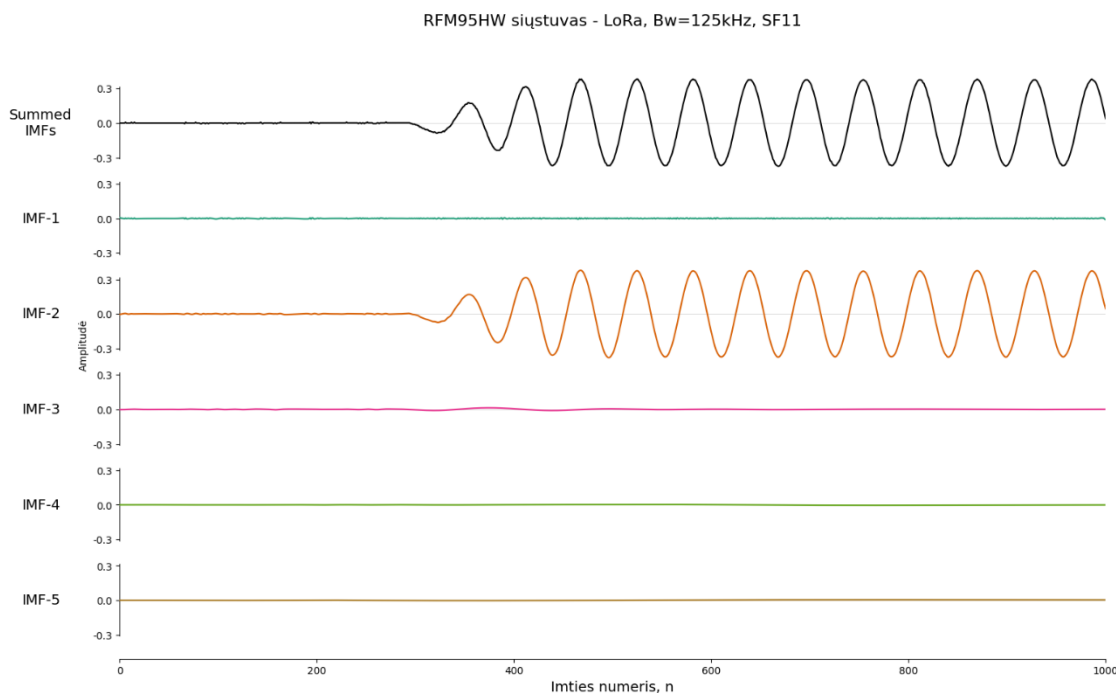
36 pav. RFM69HW RS Hilberto-Huango GFSK moduliacijos spektras, kai poslinkio dažnis yra 5 kHz, kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas

Paskutinis analizuojamas RS yra RFM95HW. Pateikiamos atskiros IMF komponentės – 500 kHz (žr. 37 pav.) ir 125 kHz pločių (žr. 38 pav.). Analizuojant 500 kHz plotį, matyti, jog pagrindinis signalas didžiąja dalimi susideda iš IMF–1 bei turi šiek tiek nutekintos energijos į IMF–2, tačiau tai nėra pereinamasis signalas.



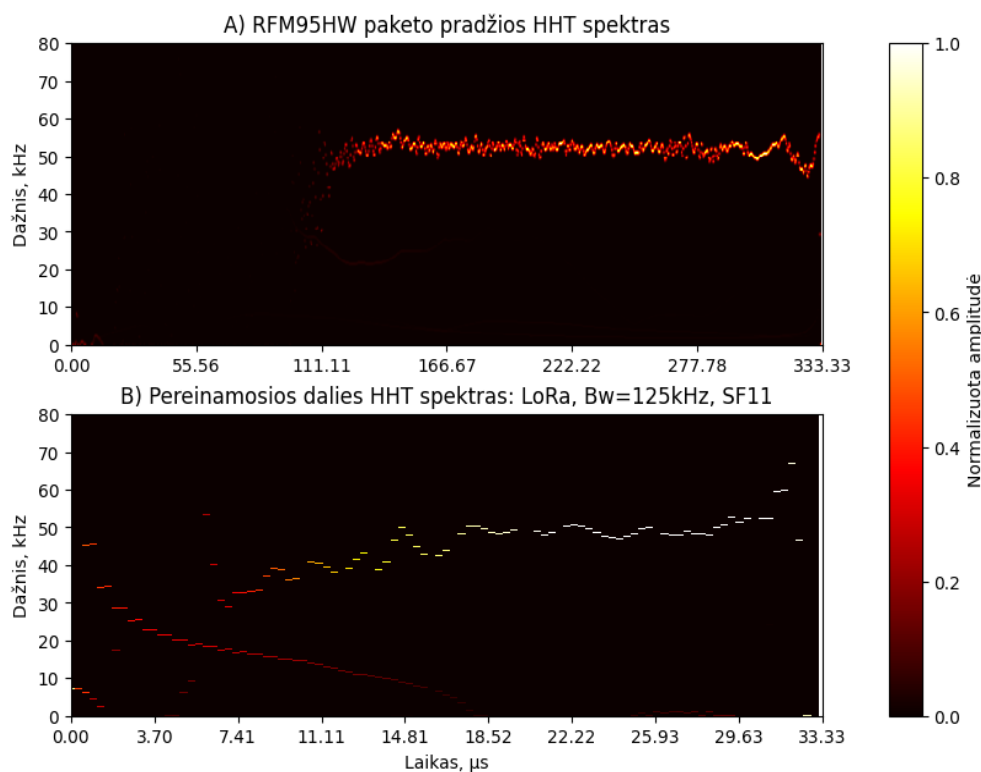
37 pav. RFM95HW radijo siųstuvo duomenų paketo IMF, kai plotis 500 kHz

Esant mažesniai pločiui, visa energija atitenka IMF-1, o kitos komponentės jokios naudingos informacijos neneša, išskyrus IMF-3, kuri turi labai nežymų amplitudės kitimą ties 400 imties tašku.



38 pav. RFM95HW radijo siųstuvo duomenų paketo IMF, kai plotis 125 kHz

Pateikiama RFM95HW RS keletas skirtingų HHT spektrų – vienas naudoja 125 kHz plotį, o kitas – 31.25 kHz. Analizuojant 39 pav. esančius RFM95HW RS spektrus, matyti, jog A dalyje yra matoma gana stabili signalo pradžios energija ties 50 kHz, tačiau artėjant 333,33 μ s, ji pradėjo virpėti su didesniu amplitudės pokyčiu. Pereinamojo signalo dalyje (B dalis) matomos kelios energijos trajektorijos, tačiau, atsižvelgiant į normalizuotą amplitudę, yra tik viena pagrindinė trajektorija (didžiausias energijos kiekis) nuo 5,26 iki 31,58 mikro–sekundės. Šios trajektorijos dažnis stabiliai kyla nuo 30 kHz (nežymus energijos kiekis) iki 60 kHz (didžiausias energijos kiekis)

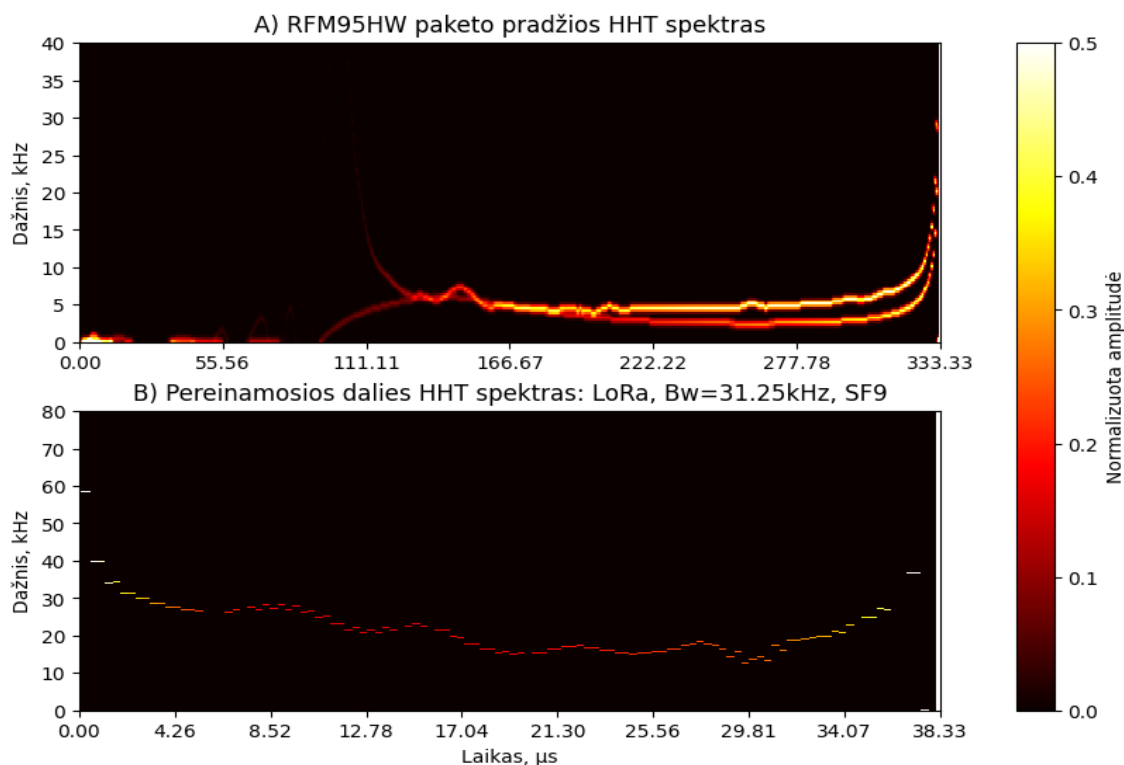


39 pav. RFM95HW RS Hilberto-Huango LoRa moduliacijos spektras (kai plotis – 125 kHz), kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas

Analizuojant mažesnio pločio signalo (žr. 40 pav.) A dalį, nuo 0 iki 95,24 μ s stebimas nežymus energijos kiekis ties 5 kHz, tačiau nuo 119,05 μ s iki pat 333,33 μ s, matomas energijos kiekio padidėjimas ties 10 kHz. B dalyje, matomas gana pastovus signalo energijos pokytis nuo 0 iki 38,33 μ s, kuris prasideda ties 60 kHz ir baigiasi ties 40 kHz.

Žvelgiant į pilną duomenų paketo signalą, matyti, jog jo visos dažninės komponentės siekia iki 200 kHz, tačiau didžiausia energijos koncentracija yra iki 100 kHz. Pateikiami du pereinamųjų signalų Hilberto spektrai – normalizuotos amplitudės ir paryškintas spektras, jeigu energijos kiekis yra daugiau nei 0. Fiksuojamas energijos kiekio pasiskirstymo kitimas laike – nuo 0 iki 3,51 μ s energija pereina iš 0 Hz iki 60 kHz ir iki pabaigos išlieka gana pastovi, tačiau su didesne energijos koncentracija.

Išanalizavus skirtingų RS signalų pradžias ir įsitikinus, jog pagal juos atskirti RS yra įmanoma, kita užduotis yra sukurti šių RS duomenų biblioteką, kurią naudojant, bus apmokomi neuroniniai tinklai.



40 pav. RFM95HW RS Hilberto-Huango LoRa moduliacijos spektras (kai plotis – 31,25 kHz), kur A – pateikiama išsiųsto paketo pradžia, o B – paketo priartintas pereinamasis signalas

CNN tinklo įvestims bus naudojami 256x256 dydžio paveikslai, kurių X ašis – dažnio komponentės, o Y ašis – laikas. Laiko ašis visų pereinamųjų signalų yra skirtinga, tačiau dėl vienodų duomenų matmenų pateikimo buvo suvienodintos visos įvestys, jas papildant 0 laiko srityje. Šie papildomi 0 galutiniam rezultatui jokios įtakos neturi, nes sąsūkos skaičiavimo metu jie išliks nuliniai ir joks požymis nebus ištraukiamas, tačiau tai daro neigiamą įtaką paties tinklo apsimokymo greitaveikoje bei didina paslėptų sluoksnių parametrų skaičių ir atitinkamai didėja atminties reikalavimai. Dėl paprastumo, neuroninių tinklų įvestims naudojamas vien tik pilkos spalvos diapazonas (angl. *Grayscale*), kurių reikšmės apibrėžtos nuo 0 iki 1. Atvaizdavimo tikslais, laiko sritis yra apibrėžta ir vienoda vienam siųstuvui, tad, jeigu pereinamasis signalas yra trumpesnis nei limitas, pikselių reikšmėms atvaizduoti naudojama *hot*¹² spalvų paletė.

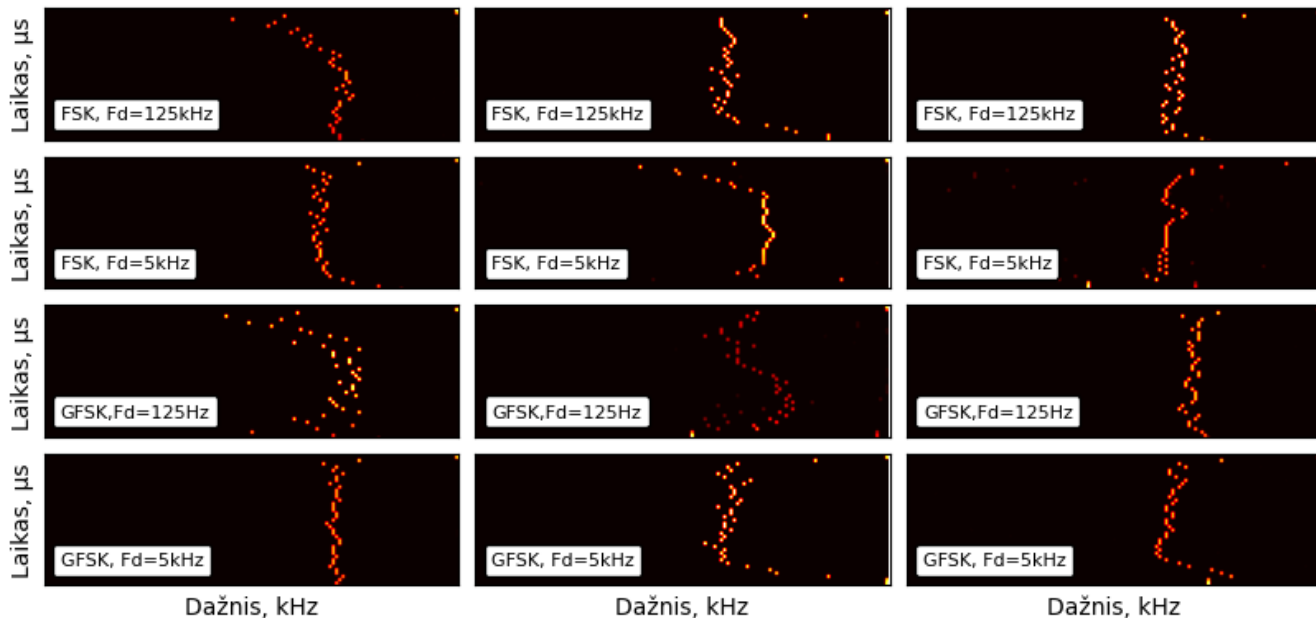
Pavyzdiniai Hilberto-Huango spektrai pateikiami 42 pav., 41 pav. ir 43 pav., kuriuose matyti, jog esant skirtingoms moduliacijoms, pereinamasis signalas yra skirtingas, nors tai yra to paties RS pereinamieji signalai, tad iš šių duomenų galima daryti išvadą, jog norint teisingai apmokyti tinkle, reikia naudoti kuo daugiau to paties RS moduliacijos pavyzdžių. Taip pat buvo nustatyta, jog ne visi IMF yra paskaičiuojami teisingai ir tam tikra informacija gali būti prarandama, o norint to išvengti, galima prie originalaus signalo pridėti Gauso triukšmą, kuris labiau paskirstytą triukšmą. Tuomet didesnės energijos IMF būtų teisingiau ištraukiami, o pats triukšmas būtų iškompensuojamas skaičiuojant IMF.

Lyginant RFM22 ir RFM69HW siųstuvus, matyti, jog RFM22 didžiausias energijos kiekis, laiko atžvilgiu, yra išsidėstęs gana tolygiai vienoje dažnio srityje, o RFM69HW energijos kiekis kinta ir

¹² <https://matplotlib.org/stable/users/explain/colors/colormaps.html#sequential2>

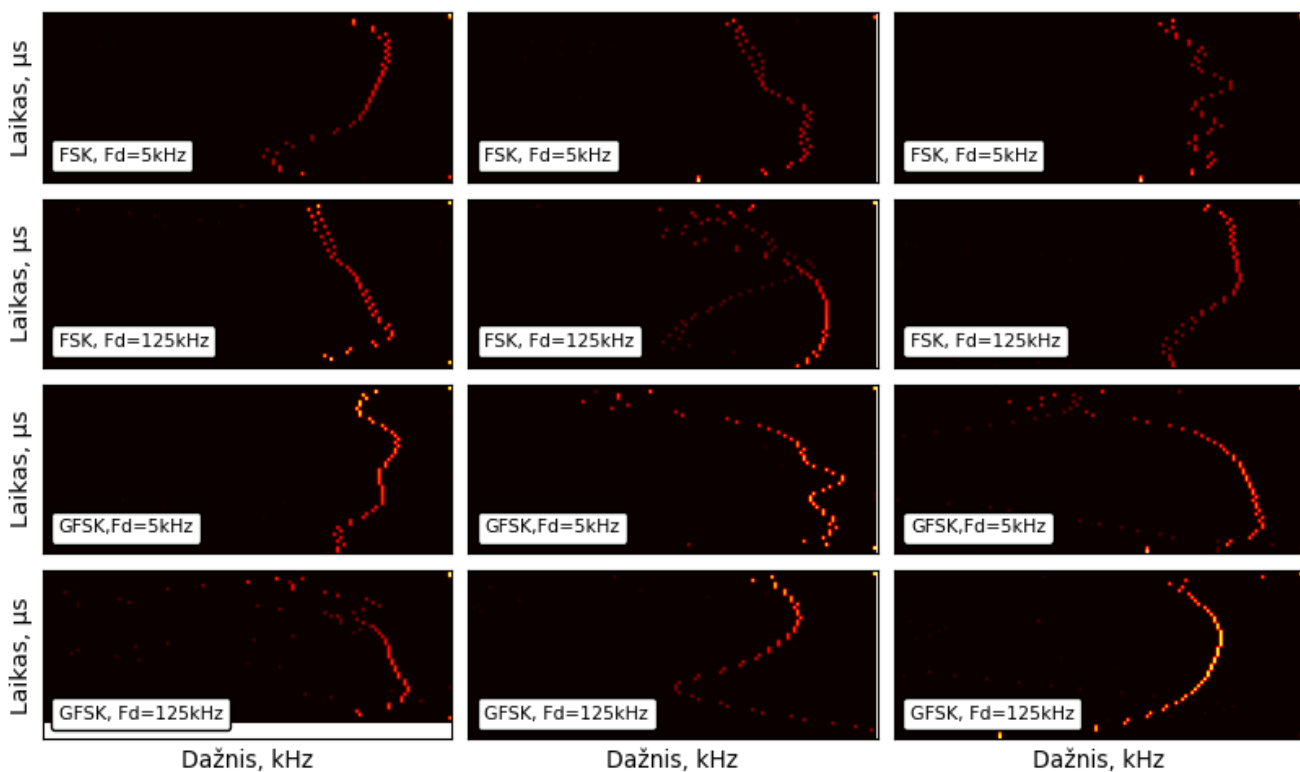
sudaro „lanką”. Kad būtų galima tiesiogiai lyginti RFM22 ir RFM69HW RS, atvaizduojamos tos pačios moduliacijos. Gana kokybiškas ir pastovus spektras yra RFM95HW siųstuvo – pateikiamos trys skirtingos moduliacijos, kurias vizualiai nesunku atskirti, o tarp tos pačios moduliacijos paketų išlaikoma gana aukšta TFED koreliacija.

RFM22 siųstuvo, originalių paketų Hilberto - Huango spektro požymiai (256 x Y pikseliai)



42 pav. RFM22 siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms

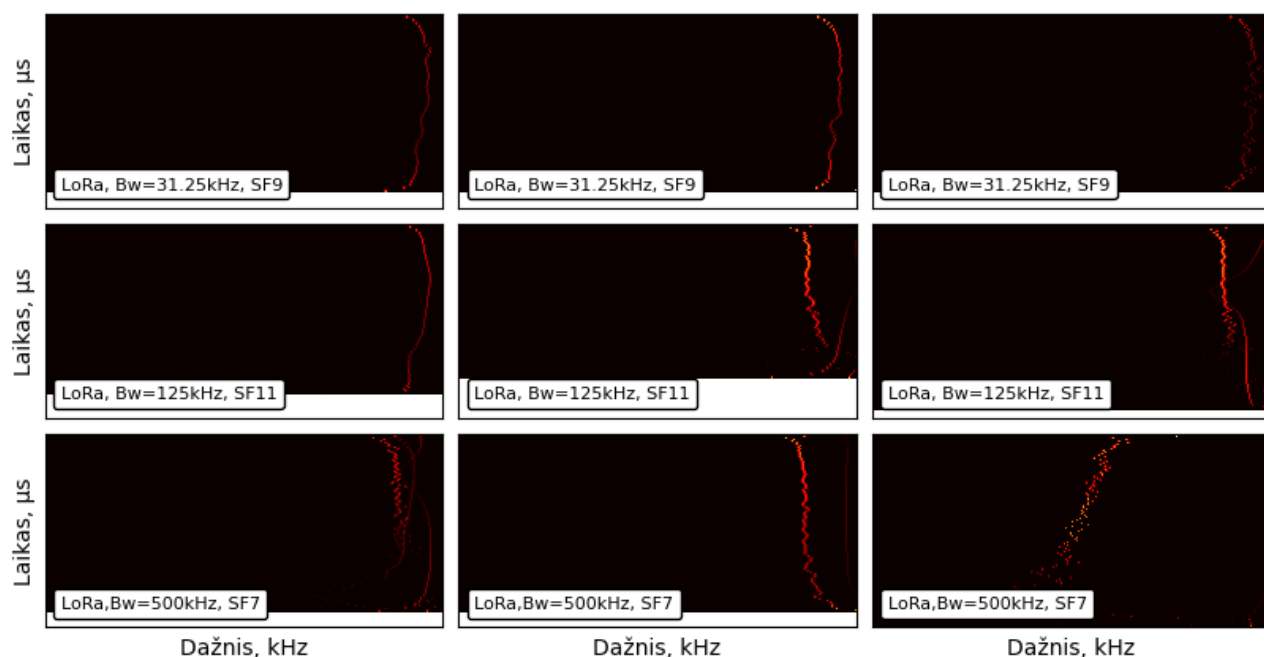
RFM69HW siųstuvo, originalių paketų Hilberto - Huango spektro požymiai (256 x Y pikseliai)



41 pav. RFM69HW siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms

Analizuojant RFM95HW RS skirtingų moduliacijų spektrus, galima išvelgti, jog tai yra gana pastovių požymių spektrai – tos pačios moduliacijos HHT yra panašios formos ir stiprumo.

RFM95HW siųstuvo, originalių paketų Hilberto - Huango spektro požymiai (256 x Y pikseliai)



43 pav. RFM95HW siųstuvo HHT spektro pavyzdžiai esant skirtingoms moduliacijoms

Taigi, atlikus išsamią pereinamųjų signalų analizę naudojant Hilberto-Huango transformaciją, buvo įsitikinta, jog kiekvienas siųstuvai turi skirtingus pereinamojo signalo požymius, kurie galėtų padėti juos išskirti ir suklasifikuoti duomenų siuntimo metu.

3.4. Pereinamųjų signalų duomenų išplėtimas

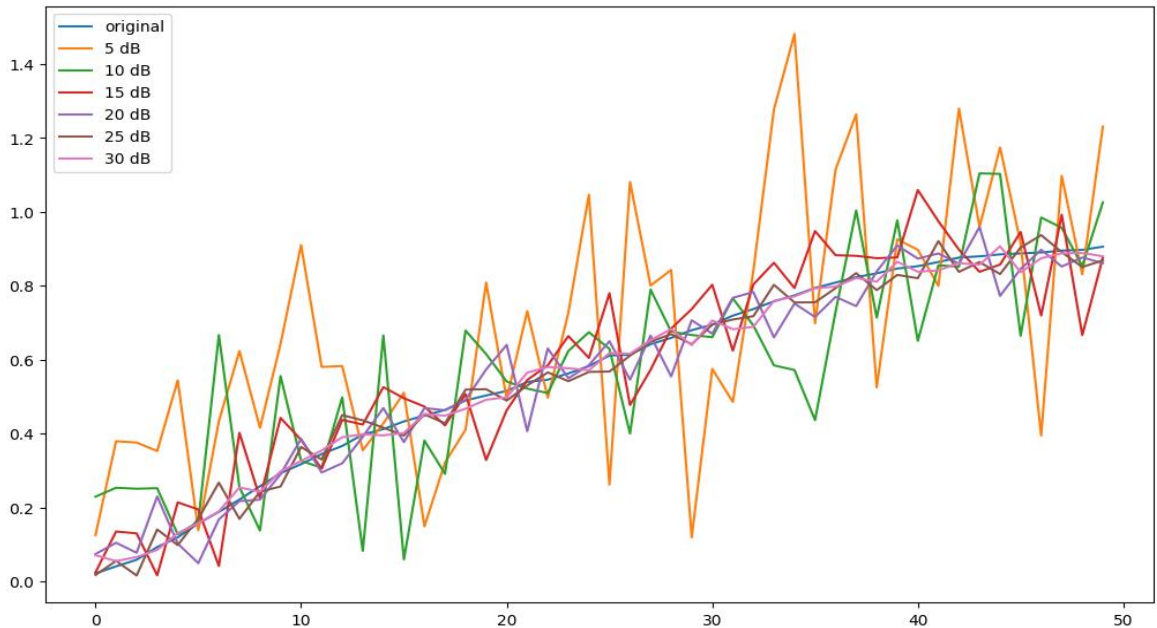
Šiame darbe apskaičiuojami požymiai neuroniniam tinklui yra tik paveikslas ir įprastai, kai įvestis būna paveikslas, duomenys yra transformuojami įvairiais būdais: rotacijos, paveikslų padidėjimas ar sumažinimas bei spalvų paletės pakeitimai. Tokios transformacijos atliekamos norint užtikrinti geresnį tinklo apmokymą, naudojant kuo didesnę kiekį tų pačių duomenų variacijų, t.y. duomenys yra išplečiami (angl. *Data augmentation*). Šio darbo duomenims rotacija ir kiti įprastiniai paveikslų transformacijos būdai nėra galimi, nes duomenys yra HHT spektras, kuris vaizduoja tikslią TFED informaciją, tačiau yra galimybė praplėsti laiko srities pereinamuosius signalus, prie jų pridėdant Gauso baltąjį triukšmą, – sugeneruojamas skirtingų SNR lygių signalas. Toks būdas imituoja triukšmingas aplinkas, o tai gerina apmokyto modelio tikslumą įvairiose aplinkose.

Prieš generuojant baltąjį triukšmą, reikia nustatyti dabartinį paketo SNR. Naudojamas laiko srities būdas, kai atskirai suskaičiuojamos to paties signalo ir triukšmo galios, o iš jų, logaritminėje skalėje, apskaičiuojamas skirtumas. Žinant bazinį SNR, galima generuoti skirtingą triukšmą, kurį sudėjus su signalu, bus gautas naujas signalas su nurodytu SNR. Pasirinkta generuoti 5, 10, 15, 20, 25 ir 30 dB SNR signalus. Kiekvieno paketo SNR paskaičiuojamas naudojant formulę¹³:

¹³ <https://github.com/hrtlacek/SNR/blob/main/SNR.ipynb> (pirmas metodas)

$$SNR_{dB} = 10 * \log_{10} \frac{P_s - P_n}{P_n} \quad (22)$$

čia P_s – signalo ir triukšmo bendra galia, o P_n – tik triukšmo galia. Triukšmas yra laikomas signalu, esančiu iki pereinamojo signalo pradžios. Generuojant AWGN (angl. *Additive white Gaussian noise*), triukšmas atsitiktinai generuojamas atskirai – realiai ir menamai dalims. Atitinkamai, generuojamo triukšmo lygis padalijamas per pusę kiekvienai daliai. Prieš pasirenkant naujas paketų SNR vertes, buvo nustatomas kiekvieno originalaus paketo SNR ir išskaičiuojamas vidutinis SNR (žr. 14 lentelė). Iš duomenų nustatytas didžiausio naudotino SNR slenkstis – 30 dB.



44 pav. Laiko srityje pateikiamo pereinamojo signalo pavyzdys esant skirtingoms SNR reikšmėms

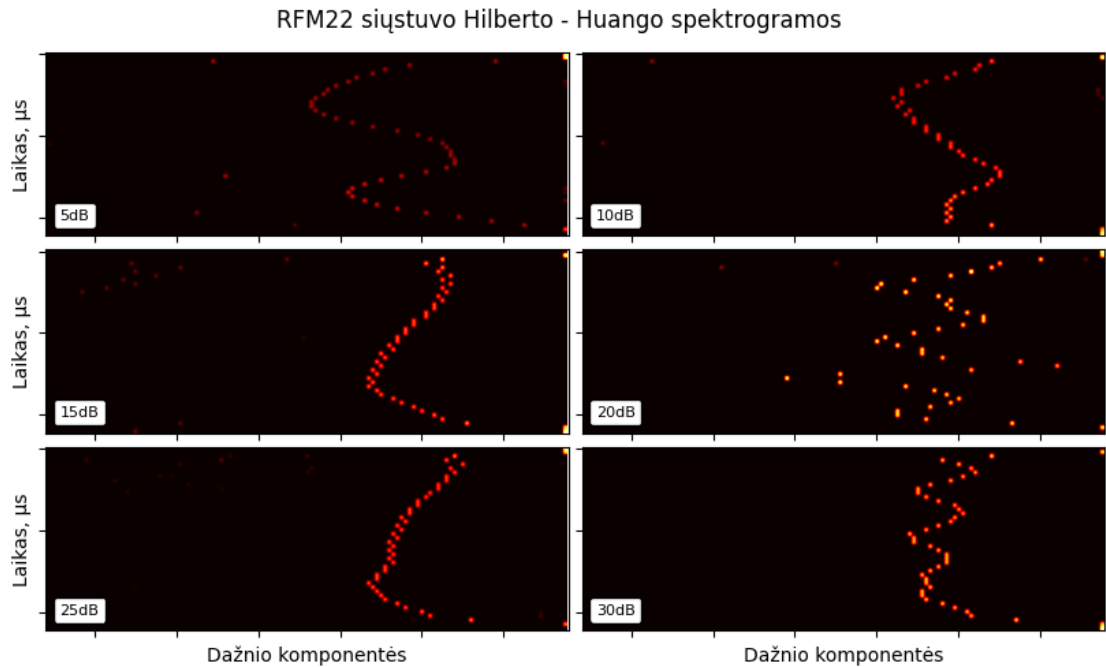
14 lentelė. Originalių duomenų apskaičiuotas vidutinis SNR, imtis – 750 paketų

Siųstuvo modelis	Moduliacija	Vidutinis paketų SNR
RFM22B	FSK	42 dB
	GFSK	46 dB
RFM69HW	FSK	31 dB
	GFSK	31 dB
RFM95W	LoRa	39 dB

Duomenų išplėtimo procesas yra vykdomas po 17 pav. pavaizduoto algoritmo, nustačius pereinamojo signalo pradžią (Žingsnis nr. 1) ir galą (Žingsnis nr. 2) originaliame signalo. Tie patys pradžios ir pabaigos imties taškai naudojami ir su AWGN išplėsto signalo pereinamosios dalies ištraukimui.

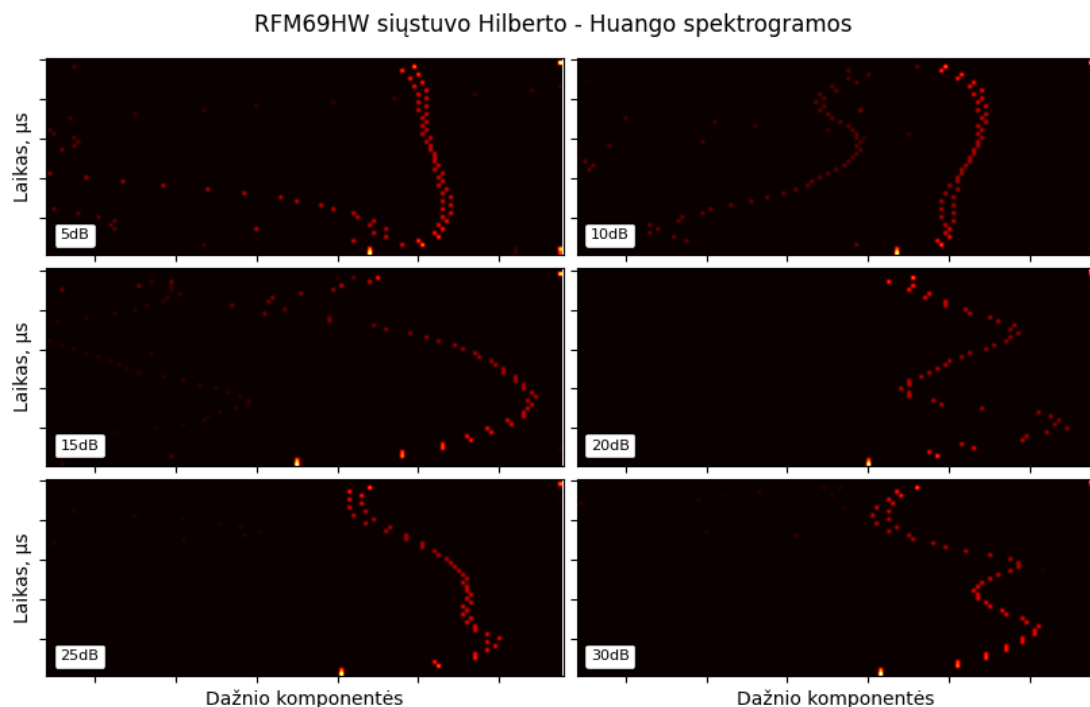
Turint minėtą pereinamąjį signalą, jis, kaip ir originalus, yra pateikiamas HHT skaičiavimams. Laiko srities atsitiktinai parinktas pavyzdinis signalas pateikiamas 44 pav., iš kurio matyti, jog mažėjant SNR, originalus signalas yra vis labiau išdardomas ir kinta jo savybės. Esant žemoms SNR reikšmėms, radijo siųstuvo atpažinimas naudojant vien tik laiko srities požymius tampa labai sudėtingas ar net nebeįmanomas.

RFM22 siųstuvo spektras 46 pav., kurio analizė parodo, jog, esant skirtingai SNR reikšmei, pereinamasis signalas, nors ir panašios formos, tačiau, kai kuriais atvejais, turi arba mažiau energijos, arba energijos dispersija yra didesnė. Pavyzdžiui, esant 5 dB SNR reikšmei, energijos kiekis yra mažesnis per visą pereinamojo signalo laiką ir, ko pasekoje, jo forma yra sunkiau matoma, o esant 30 dB SNR reikšmei, signalo forma matoma gana aiškiai – energijos kiekis yra didelis ir neišsisklaidęs.



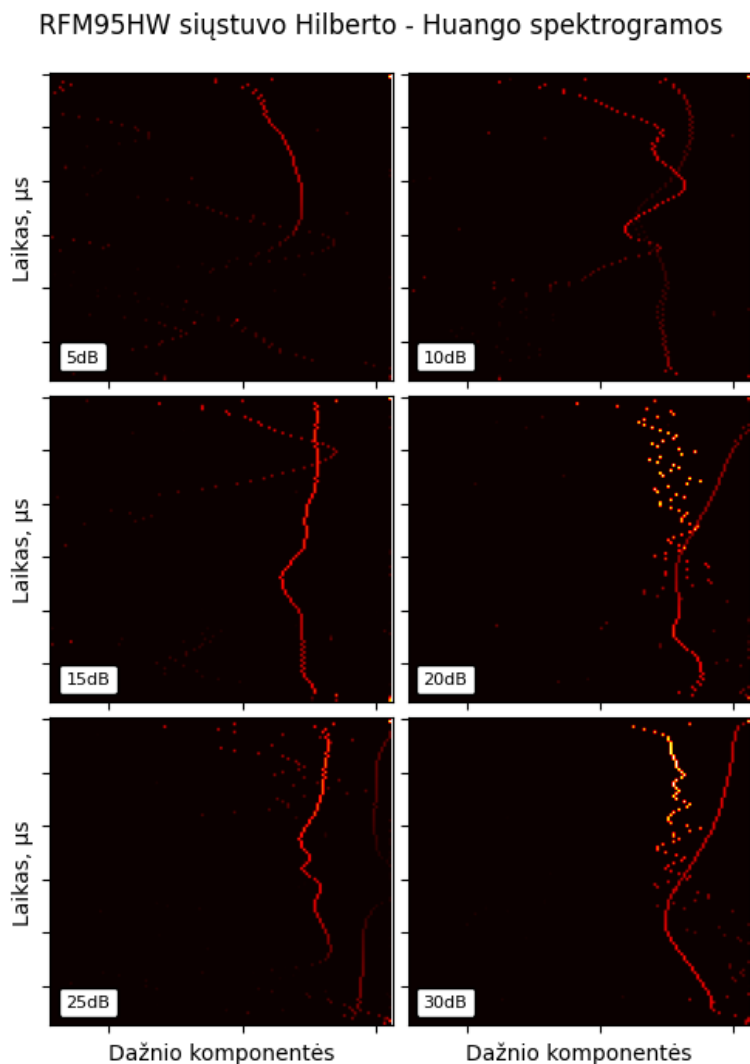
46 pav. RFM22 siųstuvo skirtingų SNR reikšmių, pereinamieji signalai

Esant didesnei SNR reikšmei, energijos kiekio koncentracija yra didelė ir pereinamojo signalo kreivė bei jos kontūrai yra aiškiai matomi, tačiau žemėjant SNR, ši energija silpsta ir prie 5 dB energijos kiekis yra vos matomas.



45 pav. RFM69HW siųstuvo, skirtingų SNR reikšmių, pereinamieji signalai

Atitinkamai, pateikiamas ir RFM69HW siųstuvo spektras (žr. 45 pav.). Išanalizavus gautus rezultatus, matyti, jog spektras tampa mažiau tolygus esant skirtingoms SNR reikšmėms. Taip pat, stebima, kad prie žemesnių SNR atsiranda ir mažos energijos antra dažnio komponentių kreivė.



47 pav. RFM95HW siųstuvo, skirtingų SNR reikšmių, pereinamieji signalai

Atlikus RFM95HW RS spektro (žr. 47 pav.) analizę, matyti, jog esant 15, 20, 25 ar 30 dB SNR reikšmėms, yra matomos dvi dažnių komponentių kreivės. Prie aukšto SNR, pereinamojo signalo kreivės yra aiškios ir matoma didelė energijos koncentracija. Žemėjant SNR reikšmei, antroji kreivė pradeda nykti, kol pasiekusi 5 dB SNR reikšmę, pastaroji kreivė beveik visiškai išnyksta, o ir pagrindinės kreivės energija tampa labai nežymi

Galima daryti prielaidą, jog dėl Hilberto transformacijos ypatybių, t.y. vidinio režimo funkcijų (IMF) panaudojimo ir nustatyto išankstinio limito, esant žemam SNR lygiui, nėra išgaunamas pilnas informacijos kiekis, tačiau didinant ištraukiamų IMF kiekį, padidinamas skaičiavimų intensyvumas bei įnešamas papildomas triukšmas į Hilberto-Huango spektrą.

Atlikus dirbtinį duomenų išplėtimą, sudaromas mokymosi duomenų rinkinys, kurio sudėtis pateikiama 15 lentelėje. Nustatyta, jog bendras duomenų rinkinio kiekis yra 5250 paketų, iš kurių po 2100 susideda iš RFM22 ir RFM69HW paketų, o likę, 1050 paketų, priklauso RFM95HW.

15 lentelė. Radijo siųstuvų išplėstų ir originalių duomenų kiekių apibendrinimas

Siųstuvas	SNR, dB	Paketų kiekis	Bendras paketų kiekis	Iš viso
RFM22	originalus	300	2100	5250
	5	300		
	10	300		
	15	300		
	20	300		
	25	300		
	30	300		
RFM69HW	originalus	300	2100	
	5	300		
	10	300		
	15	300		
	20	300		
	25	300		
	30	300		
RFM95HW	originalus	150	1050	
	5	150		
	10	150		
	15	150		
	20	150		
	25	150		
	30	150		

3.5. Duomenų analizės apibendrinimas

Analizės metu buvo apžvelgti niekaip neapdoroti I/Q duomenys, iš kurių išgautas pereinamasis signalas paverčiamas į laiko – dažnio – energijos sritį naudojant STFT ir Hilberto-Huango transformacijos metodus. Padaryta išvada, jog STFT metodo nepakanka norint tiksliai išnagrinėti pereinamuosius signalus, kurie trunka vos keletą mikro – sekundžių. Naudojant Hilberto-Huango transformacijos metodą, išanalizuoti trijų skirtingų RS pereinamieji signalai esant skirtingoms moduliacijoms – FSK, GFSK ir LoRa. Pastebėta, jog skirtumai tarp pačių RS, nors ir naudojamos tos pačios moduliacijos, yra gana dideli, o tai leidžia daryti prielaidą, jog RS atskyrimas yra įmanomas. Taip pat, tarp to pačio RS ir jo pasirinktos, vienodos moduliacijos, galima įžvelgti skirtumų, tad norint kuo teisingiau apmokyti modelius, yra reikalingas didelis kiekis duomenų.

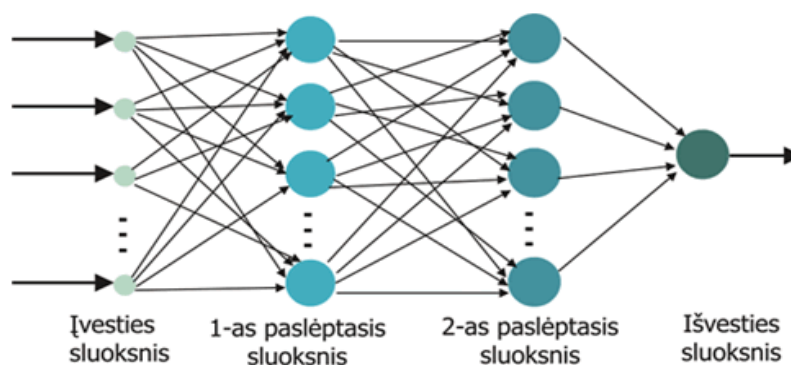
Taipogi, atsižvelgiant į literatūroje minėtus darbus, surinktų duomenų tinklo apmokymui gali nepakakti, tad signalai buvo dirbtinai išplečiami – pridėdant AWGN triukšmą, gauti skirtingų SNR pereinamieji signalai, kurių bendras kiekis sudaro 5250.

4. Duomenų klasifikavimo tyrimas ir metodai

4.1. Konvoliucinis neuroninis tinklas

Šiame darbe esantis HHT spektras ypatybės vaizduoja signalų laiko – dažnio – energijos pasiskirstymą, o tai galima laikyti kaip paveikslėlio atitikmenį, kur laikas yra Y ašis, dažnis – X ašis, o energijos kiekis – vieno pikselio vertė (X, Y) ašyje. Dėl šios priežasties, klasifikavimo tyrimui atlikti, buvo parinktas CNN tinklas, kuris yra viena iš giliojo mokymosi modelių klasių. Toks pavadinimas suteiktas dėl sąsūkos sluoksnių panaudojimo, kuris geba išskirti kampus, kraštines ar spalvas, o vėlesniuose sluoksniuose šiuos požymius sujungia į sudėtingus ir tarpusavyje susietus požymius. Aukšto lygio neuroninio tinklo struktūra pateikiama 48 pav. – matyti, jog modelis turi įvesties, paslėptus bei išvesties sluoksnius.

Klasifikavimui atlikti ir rezultatams palyginti panaudojami trys skirtingi modeliai: VGG-16, ResNet ir „Flatten Free“ tinklas. Visi modeliai yra CNN tinklai, tačiau pritaikyti ir optimizuoti vaizdų atpažinimui ir klasifikavimui.



48 pav. Pavyzdinė neuroninio tinklo sandaros iliustracija [28]

Analizuojant vieną išskirtą paslėptą sluoksnį, jį dažniausiai sudaro:

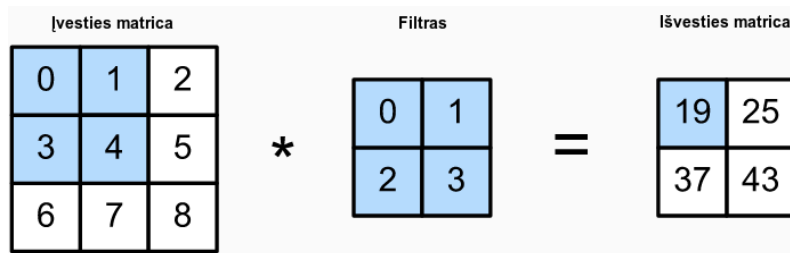
- Sąsūkos operacija – naudojant nurodyto dydžio filtrus, išskiriami požymiai;
- Aktyvacijos funkcija – išvengiama tiesinės duomenų – apsimokymo priklausomybės;
- Telkimo sluoksniai (angl. *Pooling*) – sumažinamas informacijos kiekis naudojant tam tikros matricos vidurkį arba maksimalią reikšmę.

Sąsūkos operacijos sluoksnis, turintis $N \times N$ neuronų, naudojamas $m \times m$ matmenų filtras ω , tuomet išvestis bus $(N - m + 1) \times (N - m + 1)$ dydžio, dabartinė neurolo reikšmė paskaičiuojama iš prieš tai buvusių sluoksnių naudojant sudėties ir daugybos operacijas pagal formulę¹⁴:

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{l-1} \quad (23)$$

čia x_{ij}^l – l sluoksnyje ir ij matricos pozicijoje esančio nario išvestis. Šio veiksmo iliustracija pateikiama 49 pav..

¹⁴ <https://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/>

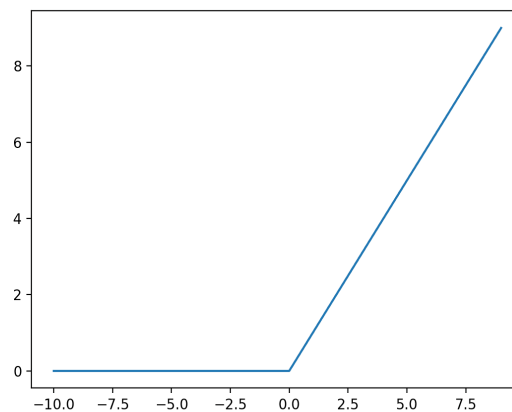


49 pav. Šašūkos operacijos pavyzdys [29]

Sekantis sluoksnis dažniausiai būna aktyvacijos funkcija – šiame darbe naudojama glodinta tiesinė funkcija (angl. *rectified linear unit (ReLU)*) [30]. Aktyvacijos funkcijos formulė yra gana paprasta:

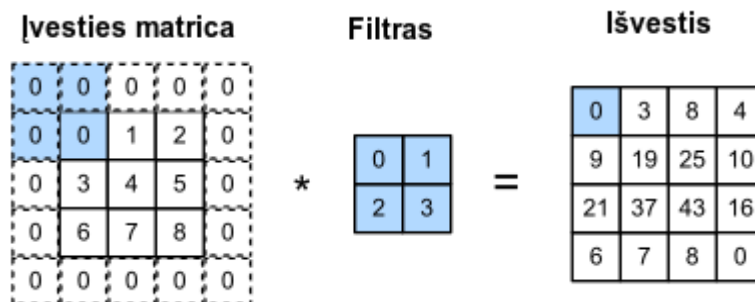
$$f(x) = \max(0, x) \tag{24}$$

ReLU yra monotoninė funkcija – jeigu įvestis neigiama, grąžinamas 0, jei teigiama, tuomet grąžinama teigiama reikšmė. Išvestis yra nuo 0 iki teigiamos begalybės. Tai yra dažniausiai naudojama funkcija neuroniniuose tinkluose dėl nesudėtingo skaičiavimo.



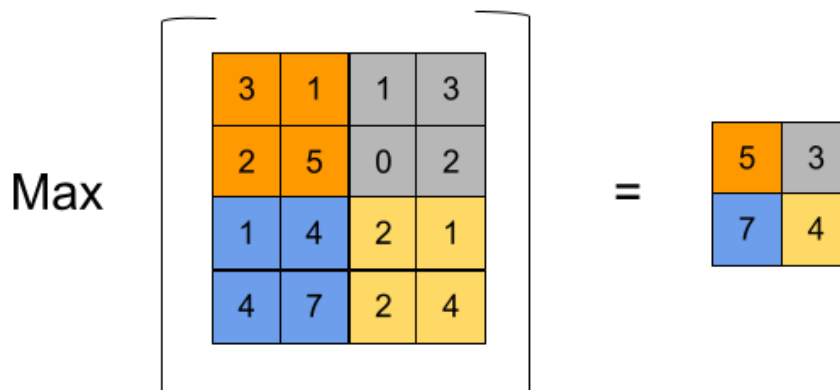
50 pav. *ReLU* aktyvacijos funkcija [30]

Atliekant šašūkos operaciją, rezultate visada gausime vis kitokio dydžio matricą. Norint valdyti gautą dydį, reikia naudoti kraštų užpildymą (angl. *Padding*). Dar vienas šašūkos operacijos valdymo parametras yra žingsnio dydis (angl. *Stride*), kuris nusako, kiek matricos elementų reikia praleisti iki kitos skaičiavimo padėties. Tokio skaičiavimo pavyzdys pateikiamas 51 pav.



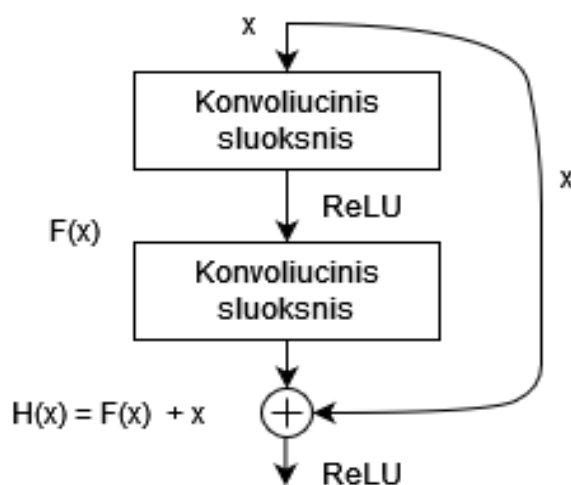
51 pav. Šašūkos operacijos ir kraštų užpildymo pavyzdys [29]

Telkimo sluoksnis naudojamas duomenų kiekio mažinimui, o būtent įvestis yra keletą kartų suspaudžiama, o tai reiškia, jog įgalinamas greitesnis duomenų apdorojimas bei mažesnis tinklas. Iliustruotas maksimalaus telkimo pavyzdys pateikiamas 52 pav. Iš pavyzdžio matyti, jog iš tam tikros laukelių grupės (pagal filtro dydį) yra išrenkama maksimali reikšmė ir patalpinama į naują matricą.



52 pav. Maksimalus telkimas (angl. *Max pooling*) [29]

ResNet tinklas naudoja liekanos blokus (angl. *Residual layer*), kur prie tolimesnio bloko išvesties yra pridama ankstesnio bloko išvestis – tokiu būdu yra išvengiama nykstančio gradiento problemos, o tai leidžia kurti daugiau sluoksnių turinčius giliuosius tinklus bei gerina apsimokymo tikslumą klasifikuojant nematytus duomenis.



53 pav. Liekanos blokas [31]

4.2. Radijo siūstuvų klasifikavimas

Giliojo mokymosi algoritmai mokosi minimizuodami klaidą tarp realios ir apmokytos reikšmės, kuri vadinama praradimo funkcija (angl. *Loss function*). Šios reikšmės mažinimui yra naudojami įvairūs optimizacijos algoritmai. Optimizacijos funkcijos darbui atlikti buvo parinktas stochastinio gradiento nusileidimo metodas (angl. *Stochastic gradient descent (SGD)*) [32], nes jis, lyginant su kitais populiariais metodais ir naudojant numatytuosius modelio parametrus, gali leisti gauti gana gerą rezultatą. Visi modeliai naudoja tą pačią mokymosi spartą – 0,001, o duomenų padavimas buvo

vykdomas duomenis išskirsčius po 64 vienetus (angl. *Batch size*). Duomenys atsitiktine tvarka išskiriami į 80 % mokymosi ir 20 % validacijos, o tai yra 4200 paketų apmokymui ir 1050 modelio testavimui. Dėl nedidelio mokymosi duomenų kiekio disbalanso, modeliui įvertinti pakaktų naudoti įprastą tikslumą, tačiau dėl konstruktyvesnių rezultatų paskaičiuojama F_1 reikšmė. Tikslumo ir F_1 reikšmių formulės pateikiamos žemiau:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (25)$$

$$F_1 = \frac{2TP}{2TP + FP + FN} \quad (26)$$

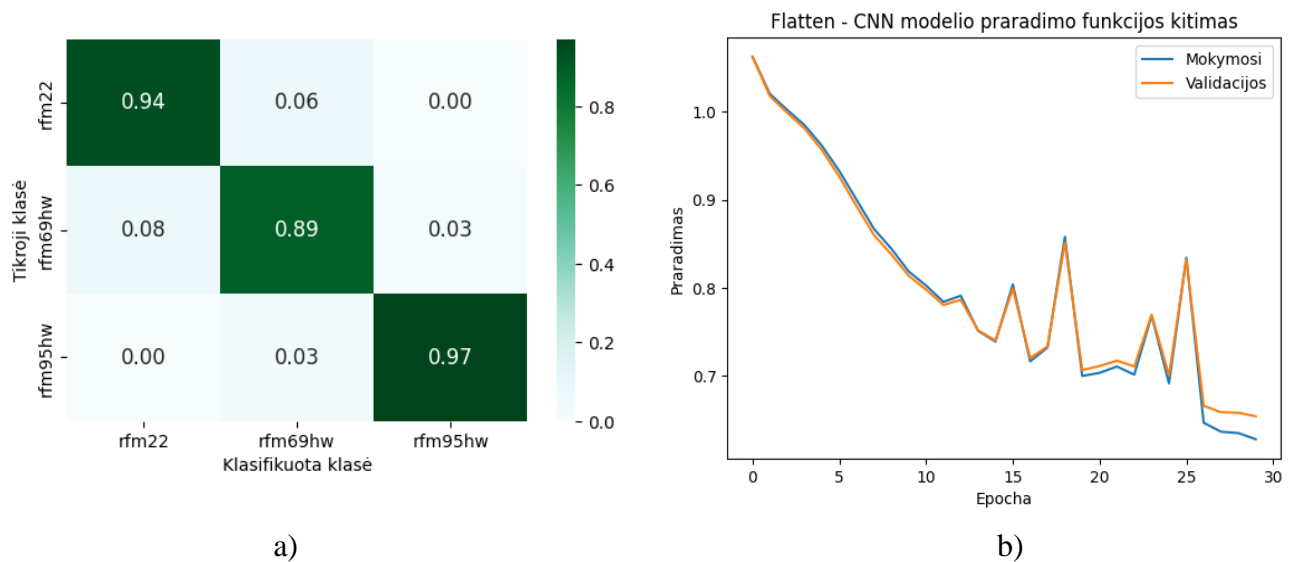
4.2.1. „Flatten Free“ CNN tinklas

Realizuotas ResNet modeliu paremtas tinklas. Remiantis 4 lentelės duomenimis, tai buvo geriausią tikslumą turintis modelis. Modelis modifikuotas, kad būtų priimti 256x256 įvesties dydžio paveikslai.

16 lentelė. „Flatten Free“ CNN tinklo modelio struktūra [20]

Sluoksniu tipas	Išvesties matmenys	Parametrų kiekis
Conv2d + BatchNorm2d + ReLU	[64, 256, 256]	1 792
MaxPool2d	[64, 65, 65]	-
Conv2d + BatchNorm2d + ReLU	[64, 65, 65]	37 056
Conv2d + BatchNorm2d + ReLU	[64, 65, 65]	37 056
ResidualBlock	[64, 65, 65]	-
Conv2d + BatchNorm2d + ReLU	[64, 65, 65]	37 056
Conv2d + BatchNorm2d + ReLU	[64, 65, 65]	37 056
ResidualBlock	[64, 65, 65]	-
Conv2d + BatchNorm2d + ReLU	[128, 33, 33]	74 112
Conv2d + BatchNorm2d	[128, 33, 33]	147 840
Conv2d + BatchNorm2d + ReLU	[128, 33, 33]	8 576
ResidualBlock	[128, 33, 33]	-
Conv2d + BatchNorm2d + ReLU	[128, 33, 33]	147 840
Conv2d + BatchNorm2d + ReLU	[128, 33, 33]	147 840
ResidualBlock	[128, 33, 33]	-
AdaptiveAvgPool2d	[128, 1, 1]	-
Flatten	128	-
Linear	3	387

Analizuojant 16 lentelėje aprašytą modelį, šis turi 676 611 tūkstančius parametrų, o jo dydis yra 146 MB. „Flatten Free“ CNN modelio apokymas truko 30 epochų, o tai yra 25,05 minutės. Klaidų matrica pateikiama 54 pav., gautas modelio tikslumas 92,76 %, o F_1 reikšmė – 0,9273.



54 pav. „Flatten Free“ CNN tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b)

Analizuojant 54 pav. pateiktą praradimo funkciją, galima daryti išvadą, jog per 30 epochų praradimas gana stabiliai mažėjo, tačiau ties 18 ir 25 epocha buvo pastebimai didesni šuoliai. Taip galėjo nutikti dėl atsitiktine tvarka pateiktų ir nuo normos ribų išeinančių duomenų.

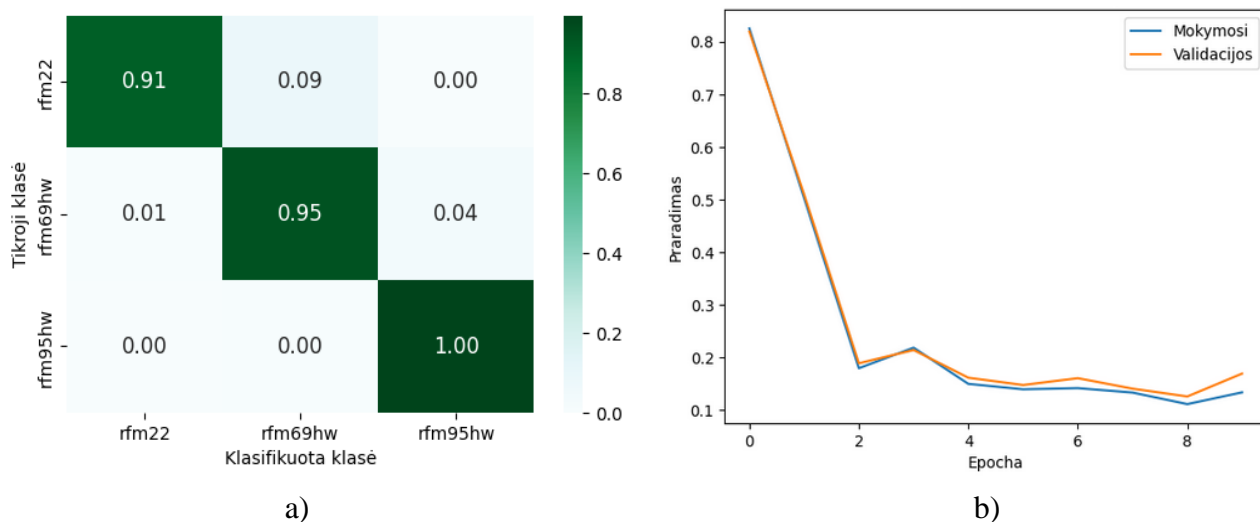
4.2.2. VGG-16 tinklas

VGG-16 modelis naudoja 13 giliuosius ir 3 pilnai sujungtus sluoksnius, visame modelyje naudojamas fiksuoto dydžio, 3x3, filtras ir vienas užpildymo elementas. Modelis buvo pasiūlytas 2013-ais metais. Pilna modelio struktūra pateikiama 17 lentelėje.

17 lentelė. VGG-16 modelio struktūra [33]

Sluoksnio tipas	Išvesties matmenys	Parametrų kiekis
2 x (Conv2d + ReLU)	[64, 256, 256]	37 568
MaxPool2d	[64, 128, 128]	-
2 x (Conv2d + ReLU)	[128, 128, 128]	221 440
MaxPool2d	[128, 64, 64]	-
3 x (Conv2d + ReLU)	[256, 64, 64]	1 475 248
MaxPool2d	[256, 32, 32]	-
3 x (Conv2d + ReLU)	[512, 32, 32]	5 899 776
MaxPool2d	[512, 16, 16]	-
3 x (Conv2d + ReLU)	[512, 16, 16]	7 079 424
MaxPool2d	[512, 8, 8]	-
Linear + ReLU	4096	2 101 248
Linear + ReLU	4096	16 781 312
Linear	3	12 291

Bendrai, VGG-16 modelis turi ~ 33.6 milijonus parametru, o jo apytikslis dydis yra 413,83 MB. Originalus VGG-16 modelis įvesties dydį numato kaip trijų kanalų (RGB) ir 224x224 dydžio paveikslą, tačiau šio darbo tikslui, kanalų kiekis buvo pakeistas iš 3 į 1, o įvesties dydis į 256x256.



55 pav. VGG-16 tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b)

Žvelgiant į šio tinklo klaidų matricą (žr. 55 pav.), matyti, jog yra žymus klasifikavimo pagerėjimas – 0.06 % tiksliau identifikuojama rfm69hw klasė, o rfm95hw klasė pagerėja per 0.03 %. Šiam tinklui apmokyti reikėjo 10-ies epochų (11,47 minutės), gautas tikslumas – 94,47 %, o F_1 reikšmė – 0,9448.

Analizuojant VGG-16 tinklo praradimo funkciją (žr. 55 pav.), matyti, jog mokymosi ir validacijos praradimo funkcijų reikšmės iki 8 epochos išliko beveik identiškos, tačiau, ties 10-ąja epocha, funkcijų reikšmės pradėjo skirtis ir didėti, o tai reiškia, jog modelis perėjo į persimokymo būseną (angl. *Overfitting*). Šios būsenos pradžioje, mokymąsi būtina stabdyti. Lyginant su „Flatten Free“ CNN tinklu, šis modelis pasiekė 1,71 % geresnį rezultatą ir 13,58 minutes greitesnį apsimokymą.

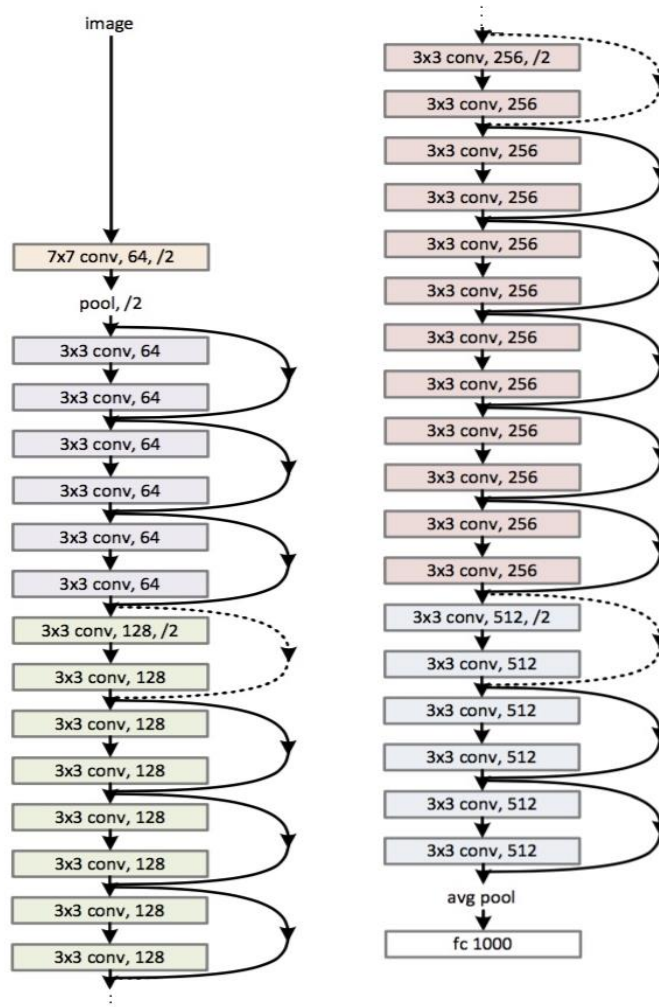
4.2.3. ResNet-34 tinklas

Kitas tinklas yra ResNet-34. Šis modelis turi 34 giliuosius sluoksnius, kurie naudoja liekanos pernešimą į sekančius sluoksnius. Modelis buvo pasiūlytas 2015 metais [34]. Šiame darbe naudojama originali ResNet-34 implementacija, kurios diagrama pateikiama 57 pav.

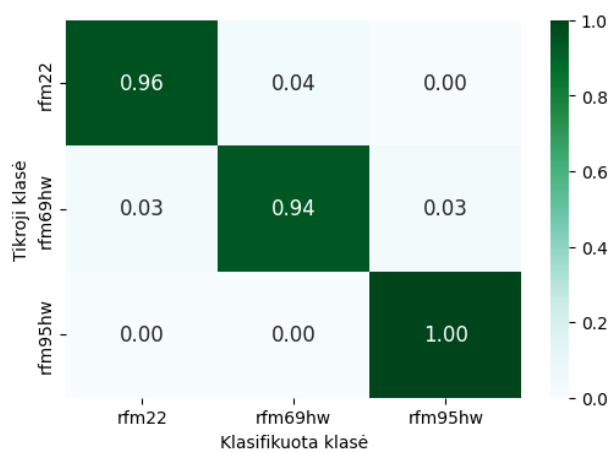
Analizuojant klaidų matricą (žr. 56 pav.), galima pastebėti, jog šis modelis, kaip ir VGG-16, turi idealų atpažinimą atskiriant rfm95hw RS klasę, o kitų klasių atpažinimas panašaus tikslumo į VGG-16 tinklo. Naudojant šį tinklą išgaunamas 99,71 % tikslumas apmokymo duomenims, validacijos duomenų tikslumas gaunamas 96 %, o F_1 reikšmė – 0,9589. Tinklui apmokyti prireikė mažiausiai epochų – 4 ir užtruko vos 3,26 minutes.

Analizuojant ResNet-34 tinklo praradimo funkciją (žr. 56 pav.), matyti, jog jau po pirmos mokymosi epochos matomas staigus praradimo funkcijos kritimas (nuo 0,8 iki 0,2), o ties 3-iaja epocha jau stebima stagnacija, tačiau, stebint validacijos praradimo funkciją ties trečia epocha, galima pastebėti tinklo persimokymą, tad didesnis epochų kiekis nebereikalingas ir net žalingas.

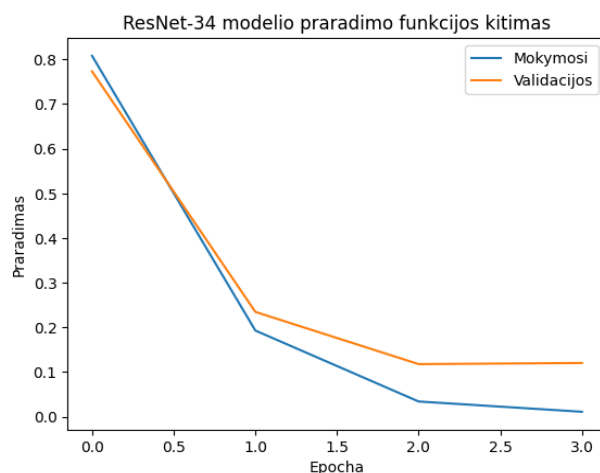
34-layer residual



57 pav. ResNet-34 modelio struktūra [34]



a)



b)

56 pav. ResNet-34 tinklo apmokymo rezultatai: klaidų matrica (a) ir nuostolio reikšmės kitimas (b)

4.2.4. Paprastas CNN tinklas

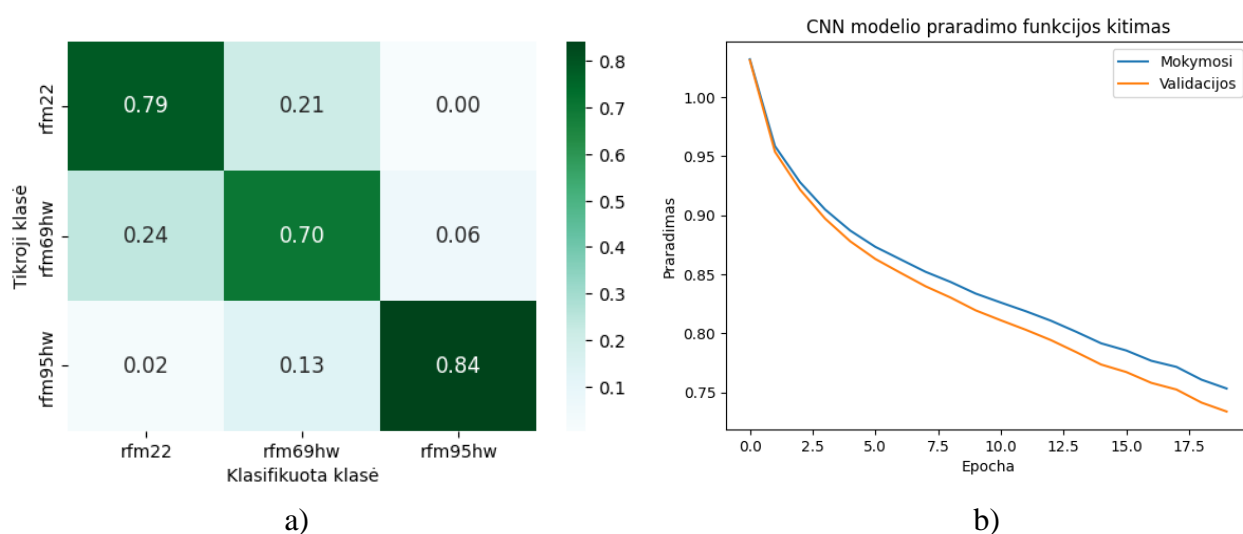
Norint sumažinti parametų kiekį, sukurtas trijų sluoksnių CNN tinklas (žr. 18 lentelė). Tinklas turi 371 331 parametrus ir jo dydis yra 183 MB. Dėl nedidelio parametų kiekio, šis modelis būtų tinkamiausias naudoti įterptinėse sistemose.

18 lentelė. Paprastas trijų sluoksnių CNN tinklas

Sluoksniu tipas	Išvesties matmenys	Parametų kiekis
Conv2d + BatchNorm2d + ReLU	[64, 256, 256]	768
MaxPool2d	[64, 129, 129]	-
Conv2d + BatchNorm2d + ReLU	[128, 129, 129]	74 112
MaxPool2d	[128, 65, 65]	-
Conv2d + BatchNorm2d + ReLU	[256, 65, 65]	295 680
AdaptiveAvgPool2d	[256, 1, 1]	-
Flatten	256	-
Linear	3	771

Išanalizavus naudotus modelius, buvo nuspręsta išanalizuoti, kokį tikslumą būtų galima išgauti naudojant paprasčiausią trijų sluoksnių CNN tinklą. Tikslas – išgauti patenkinamą klasifikavimo rezultatą naudojant kuo mažesnę parametų kiekį. Apmokius šį tinklą ir analizuojant klaidų matricą (žr. 58 pav.), gaunamas 72,36 % tikslumas mokymosi duomenų klasifikavimui, 76 % validacijos duomenų klasifikavimui, o F_1 reikšmė – 0,7646. Tinklui apmokyti prireikė 20 epochų ir užtruko 11,57 minutes.

Pagal tinklo praradimo funkcijos kreivę (žr. 58 pav.), stebima, jog praradimo funkcija mažėjo gana pastoviu greičiu iki 20 epochos, sekanciose epochose jau pradėjo vykti persimokymas, tad teko stabdyti modelio mokymąsi. Atsižvelgiant į parametų skaičių, galima teigti, jog didesnis parametų skaičius negarantuoja gero modelio rezultatų, nes paprasto CNN modelio tikslumas yra apie 16 % mažesnis.

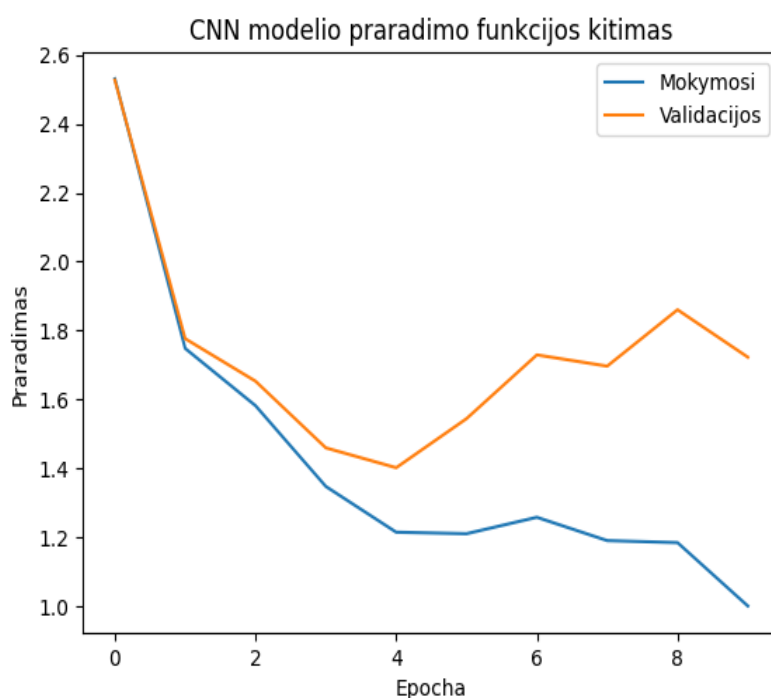


58 pav. Paprasto CNN tinklo apmokymo rezultatai: klaidų matrica (a) ir praradimo funkcijos kitimas (b)

4.3. Moduliacijos klasifikavimo tyrimas

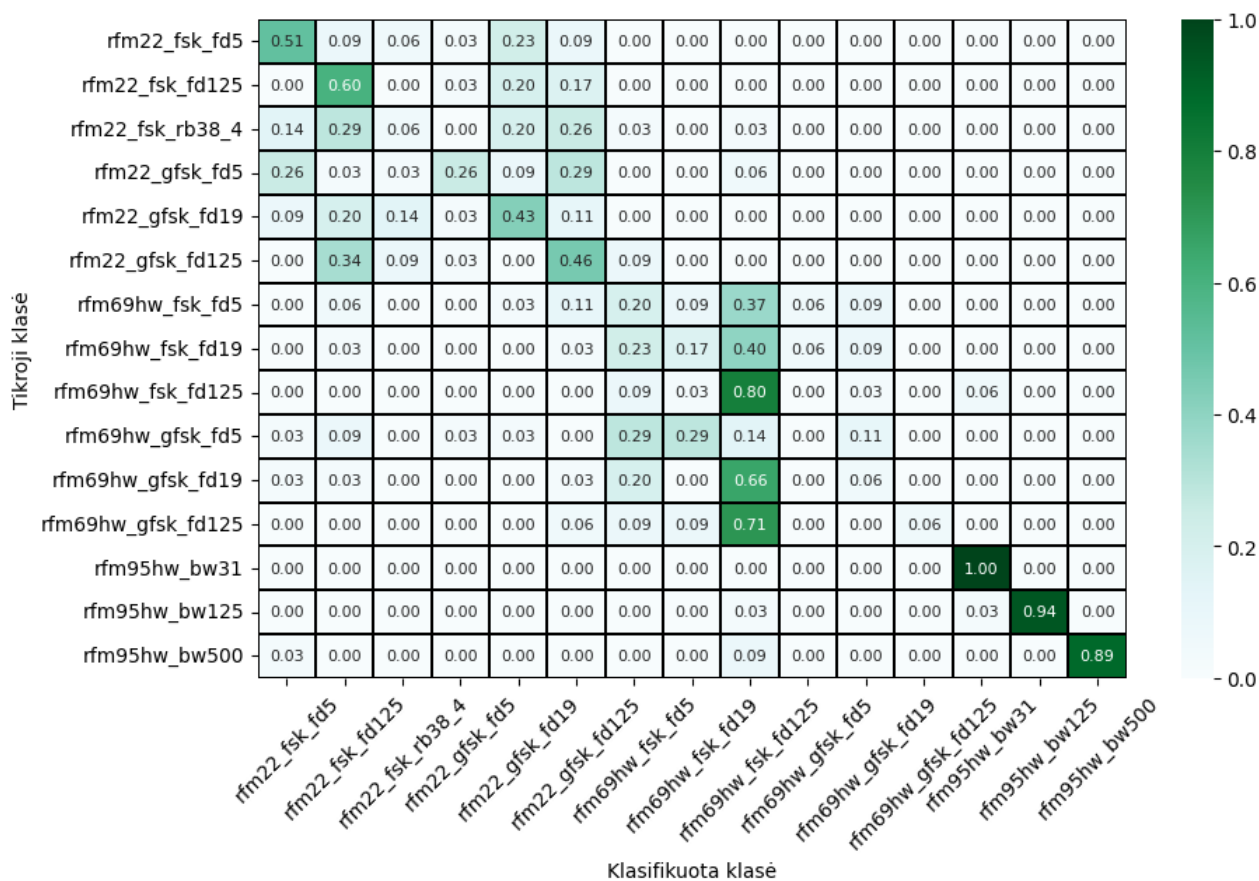
Turint surinktus ir apdorotus duomenys, tiriamas moduliacijos atpažinimo galimybės naudojant anksčiau aprašytą ResNet-34 tinklą. Moduliacijos klasifikavimui naudojama po 350 tos pačios moduliacijos paketų kiekvienam RS, o skirtingų moduliacijų kiekis yra 15.

Modelis buvo apmokomas 15 epochų, kurios truko 11,95 minutes. Su apmokymo duomenimis buvo pasiektas 57,91 % tikslumas, o klasifikuojant testavimo duomenis – 42,86 % tikslumas, F_1 reikšmė pasiekta 0,40 vertės. Mokymosi praradimo funkcijos kitimas pateikiamas 59 pav., kuriame stebima, jog jau nuo 4-os epochos tinklas pradeda būti permokomas, nes klasifikavimo rezultatai gerėja tik mokymosi duomenims, o validacijos duomenų klasifikavimo klaidos reikšmės pradeda didėti.



59 pav. Moduliacijos klasifikavimo CNN tinklo praradimo funkcijos kitimas

Taip pat, pateikiama ir klaidų matrica (žr. 60 pav.), kurioje matyti, jog ResNet-34 tinklas visiškai neatskyrė kai kurių įrenginių – *rfm69hw_gfsk_fd5* klasifikuojamas kaip *rfm22_gfsk_fd125*, *rfm69hw_fsk_fd5* ir *rfm69hw_fd19*. Atsižvelgiant į anksčiau nagrinėtą RFM22 FSK ir GFSK moduliacijų panašumą esant 5 kHz atskirties dažniui (žr. 32 pav. ir 35 pav.), modeliui pavyko gana neblogai atskirti šias dvi klases – *rfm22_fsk_fd5* klasifikavimo tikimybė siekė 0,51, o šio RS sumaišymo su *rfm22_gfsk_fd5* tikimybė siekė 0,23. Pastebėta, jog dažniausiai sumaišomos FSK ir GFSK moduliacijos. Tiksliausiai buvo klasifikuojami LoRa moduliacijų signalai – šių moduliacijų pereinamieji signalai yra ilgesni nei FSK ar GFSK moduliacijų, o ir tarp dažnių juostos pločio bei sklaidos pereinamieji signalai skiriasi ilgiu.



60 pav. Moduliacijos klasifikacijos klaidų matrica

4.4. Klasifikavimo rezultatų apibendrinimas

Analizuojant sukurtus modelius (žr. 19 lentelė), matyti, jog sudėtingiausias ir daugiausiai parametru turintis modelis yra VGG-16, o paprasčiausias – šiame darbe kurtas CNN tinklas. Iš lentelės duomenų matyti, jog parametru kiekis neturi koreliacijos su tikslumu – daug parametru turintis modelis nebūtinai turės didesnę tikslumą nei mažiau parametru turintis modelis.

19 lentelė. RS klasifikavimui naudotų neuroninių tinklų modelių apibendrinti rezultatai

Modelis	Parametru kiekis	Epochos	Galutinė praradimo reikšmė	Mokymosi laikas, minutės	Tikslumas, %	F ₁
VGG-16	33 608 387	10	0,134	11,47	94,47	0,9448
ResNet-34	21 288 451	4	0,011	3,26	99,71	0,9589
„Flatten Free“ CNN	676 611	30	0,628	25,05	92,76	0,9273
CNN	371 331	20	0,753	11,57	76,45	0,7646

Taip pat, teoriškai paskaičiuojami ir modelių dydžiai, kur VGG-16 užima 413.83 MB, ResNet-34 užima 207.22 MB, „Flatten Free“ CNN užima 146,79 MB, o paprastas CNN tinklas užima 183,43 MB. Modelio dydis priklauso nuo to, kiek neuronų yra naudojama modelyje, nes kuo didesnis naudojamų neuronų kiekis, tuo daugiau programos kintamųjų reikia saugoti atmintyje ir, atitinkamai, didėja skaičiavimų kiekis, tad ir modelio užimamas dydis auga.

Lyginant su panašaus pobūdžio darbais, šiame darbe su ResNet modeliu gautas geriausias vidutinis tikslumas yra apie 9 % geresnis nei naudojant vilnelių ir SVM metodų kombinaciją [35]. Vilnelių metodas naudojamas spektrui paskaičiuoti, o SVM metodas – įrenginiams klasifikuoti. Klasifikuojami 7 skirtingų įrenginių Bluetooth signalai ir gaunamas vidutinis 90,3 % tikslumas.

Taip pat, realizavus moduliacijos klasifikavimo modelį, nustatyta, jog gautas 42 % tikslumas nėra pakankamas modelio naudojimui, o šiame darbe naudoti pereinamieji signalai negali tinkamai atskirti RS naudojamų moduliacijos rūšių.

Išvados

1. Pasyviam skirtingų radijo siųstuvų identifikavimui pagrindiniai metodai yra mašininio mokymosi ir giliųjų neuroninių tinklų metodai. Buvo pastebėta, jog mašininio mokymosi algoritmai remiasi įvairiais statistiniais metodais, kurių rezultatai yra ypač priklausomi nuo įrenginių kiekio. Atitinkamai, naudojant giliuosius neuroninius tinklus, statistinių parametrų skaičiuoti nereikia – parametrai yra surandami apmokyto tinklo, toks būdas yra universalus ir leidžia nesunkiai pridėti naujų įrenginių į sistemą – analizuotų autorių rezultatai neretai būdavo geresni naudojant giliuosius neuroninius tinklus.
2. Išanalizavus pereinamųjų signalų išgavimo būdus, buvo nustatyta, jog EC- α būdas yra pakankamai tikslus tiriamajam darbui atlikti, tačiau norint šį metodą panaudoti realiuose sistemose, būtina atlikti tyrimus, kaip šis metodas reaguoja į SNR pokyčius. Taip pat, buvo pastebėta, jog mažėjant signalo pločiams, šis metodas praranda tikslumą – šį metodą analizuojantys autoriai to galėjo nepastebėti, nes jie analizavo signalus, kurių pločiai yra MHz intervale.
3. Originaliems duomenims atliktas dirbtinis duomenų išplėtimas – prie esamų paketų, laiko srityje, pridamas AWGN, kuriuo reguliuojamas paketo SNR ir taip išgaunamas pereinamasis signalas tampa sunkiai atskiriamas analizuojant tik laiko srities duomenis, dėl šios priežasties iškyla būtinybė laiko sritį transformuoti į dažnių srities duomenis, kurie yra atsparesni triukšmui.
4. Hilberto-Huango transformacijos metodas pateikia daugiau ir tikslesnės informacijos apie spektro komponentus. Signalų analizei atlikti panaudotas 3 MHz diskretizavimo dažnis. Analizuojant šiuo metodu apskaičiuotais RS spektrus, buvo nustatyta, jog skirtingi RS turi skirtingo ilgio pereinamuosius signalus, o tos pačios moduliacijos ir poslinkio dažnių signalai gali būti skirtingi tarp pačių RS. Taipogi, skirtumai matomi ir tarp to pačio RS ir naudojamos tos pačios moduliacijos, o tai reiškia, jog norint išgauti kuo didesnę tikslumą, reikia didelio kiekio surinktų duomenų.
5. Naudojant giliuosius neuroninius tinklus ir 5250 pereinamųjų signalų, pavyko pasiekti 99 % tikslumą klasifikuojant tris skirtingus RS – tikslumas gautas naudojant ResNet-34 modelį. Naudojant apie 5 kartus mažiau parametrų reikalaujantį „Flatten Free“ CNN tinklą, pavyko išgauti 92 % tikslumą.
6. Apibendrinant atlikto darbo rezultatus, galima teigti, jog RS identifikavimui naudojant tik pereinamuosius signalus ir Hilberto-Huango transformaciją, minimalus RS klasifikavimas yra įmanomas, tačiau norint įsitikti šio metodo tikslumu, būtina į tyrimą įtraukti didesnę kiekį RS. Taipogi, šiame darbe nebuvo nagrinėta, kokia įtaka yra daroma pereinamiesiems signalams, kai kinta aplinkos temperatūra, o tai yra svarbu, nes didžioji dalis modernių RS naudoja įtampa valdomus osciliatorius, kurie yra jautrūs temperatūros pokyčiams.

Literatūros sąrašas

- [1] Federal Communications Commission, „Radio Spectrum Allocation“, 9 July 2021. [Tinkle]. Prieiga internete: <https://www.fcc.gov/engineering-technology/policy-and-rules-division/general/radio-spectrum-allocation>.
- [2] Phelps, C. I., ir Buehrer, R. M., „Signal classification by probabilistic reasoning“, *2013 IEEE Radio and Wireless Symposium*. <https://doi.org/10.1109/rws.2013.6486672>
- [3] J. L. Xu, W. Su ir M. Zhou, „Likelihood function-based modulation classification in bandwidth-constrained sensor networks“, 2010 International Conference on Networking, Sensing and Control (ICNSC), pp. 530-533, 2010.
- [4] W. Zhang, M. Feng, M. Krunz ir A. H. Y. Abyaneh, „Signal Detection and Classification in Shared Spectrum: A Deep Learning Approach“, *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1-10, 2021.
- [5] [27] X. Li, F. Dong, S. Zhou ir W. Guo, „A Survey on Deep Learning Techniques in Wireless Signal Recognition“, *Wireless Communications and Mobile Computing*, pp. 1-12, 2019.
- [6] Y. Zeng, M. Zhang, F. Han, Y. Gong ir J. Zhang, „Spectrum Analysis and Convolutional Neural Network for Automatic Modulation Recognition“, *IEEE Wireless Communications Letters*, p. 929–932, 2019.
- [7] C. A. Hornbuckle. United States of America Patentas US7791415B2, 2010.
- [8] S.-Y. Lee, C.-H. Cheng, M. Huang ir S.-C. Lee, „A 1-V 2.4-GHz low-power fractional-N frequency synthesizer with sigma-delta modulator controller“, *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2811-2814, 2005.
- [9] S. Rajendran ir Z. Sun, „RF impairment Model-Based IoT Physical-Layer identification for enhanced domain generalization“, *IEEE Transactions on Information Forensics and Security*, t. 17, pp. 1285-1299, January 2022.
- [10] HopeRF Electronics, „RFM22B/23B ISM TRANSCEIVER MODULE“. [Tinkle]. Prieiga internete: <https://www.sparkfun.com/datasheets/Wireless/General/RFM22B.pdf>.
- [11] K. Tekbıyık, Ö. Akbunar, A. R. Ekti, A. Görçin ir G. K. Kurt, „Multi-Dimensional Wireless Signal Identification Based on Support Vector Machines“, *IEEE Access*, t. 7, p. 138890–138903, 2019.
- [12] I. Mohamed, Y. Dalveren ir A. Kara, „Performance assessment of transient signal detection methods and Superiority of Energy Criterion (EC) method“, *IEEE Access*, p. 115613–115620, January 2020.
- [13] D. Pengfei, S. Hong, J. Qi, L. Wang ir H. Sun, „A lightweight Transformer-Based approach of specific emitter identification for the automatic identification system“, *IEEE transactions on information forensics and security*, t. 18, pp. 2303-2317, 1 January 2023.
- [14] A. Ali, E. Uzundurukan ir A. Kara, „Assessment of features and classifiers for Bluetooth RF fingerprinting“, *IEEE Access*, t. 7, pp. 50524-50535, January 2019.
- [15] Y. Yuan, Z. Huang, W. Hao ir X. Wang, „Specific emitter identification based on Hilbert-Huang transform-based time-frequency-energy distribution features“, *Iet Communications*, t. 8, nr. 13, pp. 2404-2412, September 2014.
- [16] O. O. Medaiyese, M. Ezuma, A. P. Lauf ir İ. Güvenç, „Wavelet transform analytics for RF-based UAV detection and identification system using machine learning“, *Pervasive and Mobile Computing*, t. 82, p. 101569, 2022.

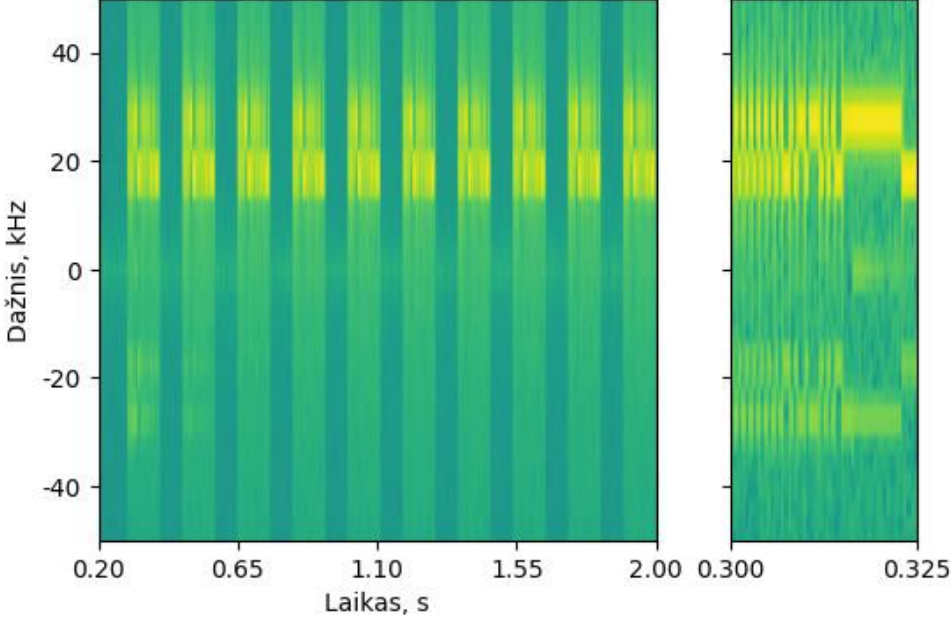
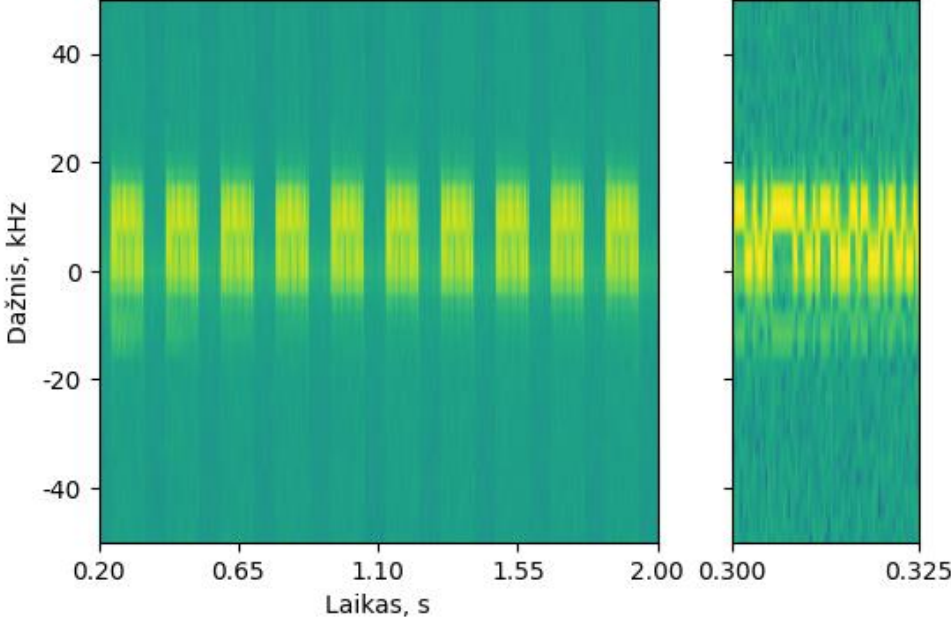
- [17] M. Ezuma, F. Erden, C. K. Anjinappa, Ö. Özdemir ir İ. Güvenç, „Detection and classification of UAVs using RF fingerprints in the presence of Wi-Fi and Bluetooth interference,“ *IEEE Open Journal of the Communications Society*, t. 1, pp. 60-76, January 2020.
- [18] G. Shen, J. Zhang, A. Marshall, L. Peng ir X. Wang, „Radio Frequency Fingerprint Identification for LoRa Using Spectrogram and CNN,“ *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1-10, 2021.
- [19] G. Shen, J. Zhang, A. Marshall, L. Peng ir X. Wang, „Radio Frequency Fingerprint Identification for LoRa Using Deep Learning,“ *IEEE Journal on Selected Areas in Communications*, t. 39, nr. 8, pp. 2604-2616, August 2021.
- [20] G. Shen, J. Zhang, A. Marshall, M. Valkama ir J. R. Cavallaro, „Toward Length-Versatile and Noise-Robust Radio Frequency Fingerprint Identification,“ *IEEE Transactions on Information Forensics and Security*, t. 18, pp. 2355-2367, January 2023
- [21] Y. Qiu, L. Peng, J. Zhang, M. Liu, H. Fu ir A. Hu, „Signal-Independent RFF identification for LTE mobile devices via ensemble deep learning,“ *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022.
- [22] J. Zhang, Q. Wang, X. Guo, X. Zheng ir D. Liu, „Radio frequency fingerprint identification based on logarithmic power Cosine spectrum,“ *IEEE Access*, t. 10, p. 79165–79179, January 2022.
- [23] I. Mohamed, Y. Dalveren, F. Ö. Çatak ir A. Kara, „On the Performance of Energy Criterion Method in Wi-Fi Transient Signal Detection,“ *Electronics*, t. 11, nr. 2, p. 269, January 2022.
- [24] P. A. A. F. W. P. C. J. M. v. d. W. a. E. F. S. P. Wagenaars, „Algorithms for Arrival Time Estimation of Partial Discharge Pulses in Cable Systems,“ *Conference Record of the 2008 IEEE International Symposium on Electrical Insulation*, pp. 694-697, 2008.
- [25] S. Markalous, S. Tenbohlen ir K. Feser, „Detection and location of partial discharges in power transformers using acoustic and electromagnetic signals,“ *IEEE Transactions on Dielectrics and Electrical Insulation*, t. 15, nr. 6, pp. 1576-1583, December 2008.
- [26] A. Milne, „The Myth of Half-wave Diversity Antenna Placement,“ [Tinkle]. Prieiga internete: <https://www.rfvenue.com/blog/2014/12/15/the-myth-of-half-wave-diversity-antenna-placemen>.
- [27] S. Taşcıoğlu, M. Köse ir Z. Telatar, „Effect of sampling rate on transient based RF fingerprinting,“ *10th International Conference on Electrical and Electronics Engineering (ELECO)*, pp. 1156-1160, 2017.
- [28] „Neuroniniai tinklai,“ [Tinkle]. Prieiga internete: <http://www.elektronika.lt/teorija/kompiuterija/4342/neuroniniai-tinklai>. [Kreiptasi balandis 2024].
- [29] „Padding and Stride,“ [Tinkle]. Prieiga internete: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html. [Kreiptasi balandis 2024].
- [30] A. F. Agarap, „Deep Learning using Rectified Linear Units (ReLU),“ *arXiv.org*, 22 March 2018.
- [31] „Residual Networks (ResNet) and ResNeXt,“ [Tinkle]. Prieiga internete: https://d2l.ai/chapter_convolutional-modern/resnet.html. [Kreiptasi balandis 2024].
- [32] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison ir G. E. Dahl, „On empirical comparisons of optimizers for deep learning,“ *arXiv (Cornell University)*, 25 9 2019.

- [33] K. Simonyan ir A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition,“ 3rd International Conference on Learning Representations (ICLR 2015) , pp. 1-14, 4 Rugsėjo 2014.
- [34] K. He, X. Zhang, S. Ren ir J. Sun, „Deep residual learning for image recognition,“ arXiv.org, 10 December 2015.
- [35] H. Almashaqbeh, Y. Dalveren and A. Kara, "A study on the performance evaluation of wavelet decomposition in transient-based radio frequency fingerprinting of Bluetooth devices," Microwave and optical technology letters, vol. 64, no. 4, pp. 643-649, 18 Sausis 2022.
- [36] P. P. S. B. P. F. R. a. T. M. Amani Al-Shawabka, „DeepLoRa: Fingerprinting LoRa Devices at Scale Through Deep Learning and Data Augmentatio,“ MobiHoc '21: Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, pp. 251-260, Liepa 2021.

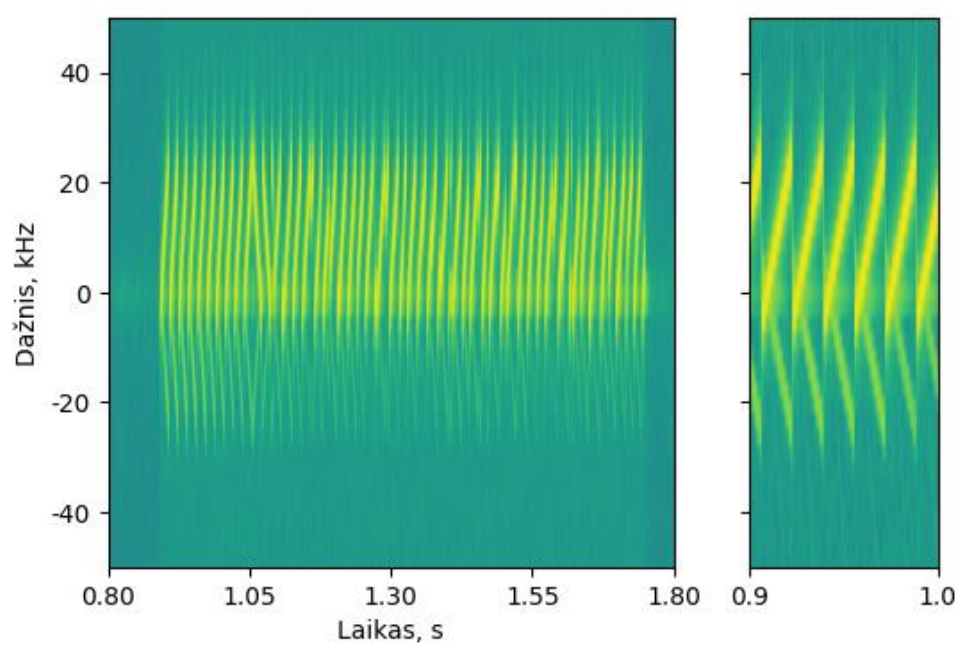
Priedai

1 priedas. Surinktų siųstuvų duomenų spektrogramos

20 lentelė. Radijo siųstuvų įrašytų paketų spektrogramos

Įrašytų duomenų aprašymas	Spektrograma
Siųstuvas: RFM22B Moduliacija: FSK Sklandos dažnis: 5 kHz	
Siųstuvas: RFM69HW Moduliacija: FSK Sklandos dažnis: 5 kHz	

Siųstuvas: RFM95HW
Moduliacija: LoRa
Plotis: 31.25 kHz
Sklaida: SF9



2 priedas. Pagalbinės funkcijos

```
import numpy as np
import ruptures as rpt

from scipy import ndimage
from scipy import signal
import json

import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.ticker as ticker
import matplotlib.gridspec as gridspec

def normalize_minmax(y, range=(0, 1)):
    y_real = np.real(y)
    y_imag = np.imag(y)

    # Calculate the standard deviation
    y_normalized_real = (y_real - np.min(y_real)) / (np.max(y_real) - np.min(y_real))
    y_normalized_imag = (y_imag - np.min(y_imag)) / (np.max(y_imag) - np.min(y_imag))

    # Scale the values: std * (max - min) + min
    y_normalized_real = y_normalized_real * (range[1] - range[0]) + range[0]
    y_normalized_imag = y_normalized_imag * (range[1] - range[0]) + range[0]

    return np.vectorize(complex)(y_normalized_real, y_normalized_imag)

def normalize_amplitude(y):
    y_real = np.real(y)
    y_imag = np.imag(y)

    y_real_max = np.max(y_real)
    y_imag_max = np.max(y_imag)

    y_real_normalized = y_real / y_real_max
    y_imag_normalized = y_imag / y_imag_max

    return np.vectorize(complex)(y_real_normalized, y_imag_normalized)

def normalize_2d_array(in_arr):
    return (in_arr - np.min(in_arr)) / (np.max(in_arr) - np.min(in_arr))

def normalize_minmax_real(y, range=(0, 1)):
    y_normalized = (y - np.min(y)) / (np.max(y) - np.min(y))

    # Scale the values: std * (max - min) + min
    y_normalized = y_normalized * (range[1] - range[0]) + range[0]

    return y_normalized
```

```

def calculate_snr(signal, noise):
    """
    Calculate a time domain SNR.

    https://github.com/hrtlacek/SNR/blob/main/SNR.ipynb (Method 1)
    """
    signal_power = np.mean(signal ** 2)
    noise_power = np.mean(noise ** 2)

    return 10 * np.log10((signal_power - noise_power) / noise_power)

def calculate_energy(y, display=False, theta=5):
    """
    Calculates the energy curve of EC-a method.
    This is used for determining the transient start.

    Args:
        y (amplitude values): A numpy list of amplitudes

    Returns:
        numpy.list: The calculated signal energy trend
    """
    signal_len = len(y)
    E = np.zeros(signal_len)

    E_signal = np.sum(y ** 2)
    sigma = E_signal / (theta * signal_len)

    E = np.cumsum(y ** 2)
    for i in range(signal_len):
        E[i] -= i * sigma

    if display == True:
        plt.figure(figsize=(20, 14))
        plt.plot(E)
        plt.show()

    return E

def extract_frame(samples, energy_curve, display=False, add_samples=100, frame_idx=0):
    """
    Given energy curve calculated using EC-a method,
    extract the frame and return a list of samples which
    start after the extracted frame.

    Args:
        samples: A numpy list of IQ signal data
        energy_curve: A numpy list of calculated energy curve
        sample_deviation: How many samples to add to start/end

    Returns:

```

```

integer: The start index of the frame (minus given add_samples value)
integer: The end index of the frame (plus given add_samples value)
"""
minimum_idx = np.argmin(energy_curve) # Paketo pradžios suradimas.
ec = energy_curve[minimum_idx:]      # Nukerpame signalo pradžią - joje galimai tik triukšmas.
end_idx = -1

start_searching = False
for i in range(len(ec) - 1):
    current_diff = ec[i+1] - ec[i]

    if current_diff > 0.1:
        start_searching = True

    if start_searching and current_diff < 0:
        end_idx = i
        break

if end_idx != -1:
    local_start_idx = 0 # Pradedame paiešką nuo minimalios reikšmės, tad pradžia yra 0.
    local_end_idx = end_idx

    if display:
        plt.figure(figsize=(20, 14))
        plt.plot(ec)
        plt.title(frame_idx)

        plt.axvline(x=local_start_idx, color='blue')
        plt.axvline(x=local_end_idx, color='red')
        plt.show()

    # Konvertuojame į globalaus signalo pradžią ir pabaigą.
    real_start_idx = (minimum_idx + local_start_idx)
    real_end_idx = (minimum_idx + local_end_idx)

    # Append some deviation to the start and end of frame.
    if (real_start_idx > add_samples):
        real_start_idx -= add_samples

    return real_start_idx, real_end_idx

return None, None

def get_peak_end_length(samples):
    avg = np.average(samples)
    max_idx = np.argmax(samples)
    tmp = samples[max_idx:]

    for i in range(len(tmp)):
        if tmp[i] < avg:
            return max_idx, i + max_idx

```

```

return None, None

def calculate_variance(samples, width=15, step=5, stop_variance=0):
    indices = np.arange(len(samples), step=5)

    # Remove the sides.
    keep = (indices >= width // 2) & (indices < len(samples) - width // 2)
    indices = indices[keep]

    var_list = list()
    for k in indices:
        start, end = k - width // 2, k + width // 2

        variance = samples[start:end].var()
        if stop_variance and variance < stop_variance:
            return [end]

        var_list.append(variance)

    return var_list

def complex_to_real(complex, fs, display=False):
    """
    Calculating complex valued EMD

    For negative part, we need a complex conjugate of
    the calculated FFT to remove the imaginary part.

    Calculate the  $X(\exp^{(j\omega)})$  and conjugate  $X(\exp^{(-j\omega)})$ 
    We get a double side spectrum.

     $X_{conjugate} = \#np.conj(np.fft.fft(np.power(-1, np.arange(len(sliced\_samples)))) * sliced\_samples)$ 
    """

    X = np.fft.fft(complex)
    X_conj = np.fft.fft(np.conj(complex))

    frequencies = np.fft.fftfreq(len(X), 1 / fs)
    X_positive = (frequencies >= 0).astype(int) * X
    X_negative = (frequencies >= 0).astype(int) * X_conj

    mean = np.mean(np.fft.ifft(X_positive + X_negative))
    x_positive = np.real(np.fft.ifft(X_positive))
    x_negative = np.real(np.fft.ifft(X_negative))

    # Might need to negate the DC offset
    real_signal = x_negative + x_positive - (mean/2)

    # Proof that x_positive and x_negative signals consist of our real `complex signal`.
    #plt.figure(figsize=(16, 12))

```

```

plt.plot(real_signal)
plt.plot(complex)
plt.legend(['Reconstructed', 'Original']); plt.show()

if display == True:
    display_FFT = X_negative #np.fft.fftshift(X_negative)

    # Calculate the power spectrum density
    f = np.linspace(-fs/2, fs/2, len(X))#np.linspace(fs/-2, fs/2, len(positiveFFT))
    #psd = np.abs(np.fft.fftshift(np.fft.fft(sliced_samples)))**2
    psd_dB = 10 * np.log10(display_FFT) ** 2

    plt.plot(f, psd_dB)
    plt.xlabel("Frequency [MHz]")
    plt.ylabel("PSD")
    plt.show()

return real_signal

def generate_awgn(total_signal, frame, desired_snr=5):
    linear_SNR = (10 ** (desired_snr / 10))

    s_power = np.mean(np.abs(frame)**2)
    n_power = s_power / linear_SNR

    noise_real = np.random.normal(0, np.sqrt(n_power/2), len(total_signal))
    noise_imag = np.random.normal(0, np.sqrt(n_power/2), len(total_signal))

    return noise_real + 1j*noise_imag

def augment_frames_varying_snr(frame, noise, print_snr=False):
    desired_snr = [5, 10, 15, 20, 25, 30]
    noise_and_frame = np.concatenate((noise, frame))
    real_snr = calculate_snr(np.abs(frame), np.abs(noise))
    frames = {}

    if print_snr:
        print(f'Clean SNR: {real_snr:.2f} dB.')

    start_idx = len(noise)
    for snr in desired_snr:
        # Don't apply AWGN if our current SNR is already less than desired!
        if int(real_snr) < snr:
            break

        awgn = generate_awgn(noise_and_frame, frame, snr)
        noise_and_frame_awgn = noise_and_frame + awgn
        noise_awgn = noise_and_frame_awgn[:start_idx]
        frame_awgn = noise_and_frame_awgn[start_idx:]

    if print_snr:

```

```
noised_snr = calculate_snr(np.abs(frame_awgn), np.abs(noise_awgn))
print(f'\tExpected SNR: {snr} | AWGN SNR = {noised_snr:.2f} dB')

frames[snr] = frame_awgn
return frames
```

3 priedas. Duomenų nuskaitymas

```
iq_files = {
    'rfm22': {
        'dir': "rfm22",
        'files': [
            "FSK_Rb125Fd125_50packets",
            "FSK_Rb2Fd5_50packets",
            "FSK_Rb38_4Fd19_6_50packets",
            "GFSK_Rb125Fd125_50packets",
            "GFSK_Rb19_2Fd9_6_50packets",
            "GFSK_Rb2Fd5_50packets"
        ]
    },
    'rfm69hw': {
        'dir': "rfm69hw",
        'files': [
            "FSK_Rb2Fd5_50packets",
            "FSK_Rb9_6Fd19_2_50packets",
            "FSK_Rb125Fd125_50packets",
            "GFSK_Rb2Fd5_50packets",
            "GFSK_Rb9_6Fd19_2_50packets",
            "GFSK_Rb125Fd125_50packets"
        ]
    },
    'rfm95hw': {
        'dir': "rfm95",
        'files': [
            "Bw31_25Cr485f512_50packets",
            "Bw125Cr455f2048_50packets",
            "Bw500Cr455f128_50packets"
        ]
    },
}

handpicked_modulation = [ ]#[ 'FSK_Rb125Fd125_50packets', 'Bw500Cr455f128_50packets' ]
handpicked_transmitter = "rfm95hw"

def show_spectrogram(name, samples, fs):
    scale = 3e3 # kHz
    ticks = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale))

    fig, ax = plt.subplots(figsize=(16, 12))
    ax.spectrogram(samples, 1024, fs, noverlap=16, scale='dB')
    ax.yaxis.set_major_formatter(ticks)
    ax.xaxis.set_major_locator(plt.LinearLocator(numticks=10))
    ax.set(ylabel='Dažnis, kHz', xlabel='Laikas, s', title=name)
    #fig.colorbar(cmap)
    plt.show()

path_head = "/home/aretasp/Documents/Magistrinis/jupyter/siustuvai"
iq_samples = {} # A dictionary for keeping the extracted I/Q samples.
```



```

display_spectrogram = False

for key, value in iq_files.items():
    dir = value['dir']
    files = value['files']
    single_transmitter = {}

    print(f'Reading IQ files of {key} module..')

    if key != handpicked_transmitter:
        print('skipping..')
        continue

    # All measurements are made 20 cm from the transmitter.
    for file in files:
        iq_file = "{0}/{1}/{2}.sigmf-data".format(path_head, dir, file)
        meta = "{0}/{1}/{2}.sigmf-meta".format(path_head, dir, file)

        if len(handpicked_modulation) > 0 and file not in handpicked_modulation:
            continue

        with open(meta, "r") as f:
            md = json.loads(f.read())
            print(json.dumps(md, indent = 3))

        fs = int(md["global"]["core:sample_rate"])
        T = 1 / fs

        ms_to_extract = 100000 # ms
        samples_in_ms = (ms_to_extract / 1000) * fs
        samples_in_100ms = int((100 / 1000) * fs)

        samples = np.fromfile(iq_file, dtype=np.complex64, count=int(samples_in_ms))
        print('File: %s\nDefined/read samples: %u/%u. The samples describe time interval of: %.2f ms'
              % (iq_file, samples_in_ms, len(samples), (T * len(samples) * 10e2)))

        if display_spectrogram:
            show_spectrogram(md["global"]["core:description"], samples, fs)

    # After decimation, the sampling rate is now 1 MHz, which is enough to cover our all modulations.
    #samples = signal.decimate(samples, 2, ftype='fir', zero_phase=True)
    single_transmitter[file] = samples

iq_samples[key] = single_transmitter # Sudedame nuskaitytus duomenis į vieną bendrą žodyną.

```

4 priedas. Pereinamojo signalo išgavimas iš pilno signalo

```
from matplotlib.ticker import FormatStrFormatter
import math

# For each transmitter, need to collect:
#
# 1. Full frame;
# 2. Starting part of the frame;
# 3. Transient:
#   3.1. Original;
#   3.2. Augmented with reduced SNR by 5, 10, 15, 20, 25, 30, 35 and 40.
training_data = {}

def extract_frames_and_transients(samples, fs, modulation, display_transient=False, display_frames=False,
display_extraction=False):
    T = 1 / fs
    idx = 0
    var_window = 20
    var_step = 5 # Dispersijos skaičiavimo žingsnis - kuo didesnis,
                 # tuo greičiau paskaičiuojama, bet didėja paklaida.

    processed_frames = list()
    number_of_samples = len(samples)
    time = T * number_of_samples

    samples = normalize_amplitude(samples)
    signal_magnitude = np.abs(samples)

    print(f'\tModuliacija: {modulation}, imties taškų kiekis: {number_of_samples}. Pateikiamas laiko
intervalas yra: {(time * 10e2):.2f} ms')
    E = calculate_energy(np.abs(samples), display=False, theta=20)

    remaining_samples = samples
    remaining_energy = E[:np.argmax(E)+100]
    add_samples = var_window + 300
    del E

    while True:
        transient_start_idx = {}
        transient_end_idx = {}

        # Su kiekviena iteracija, ištraukiamas vis naujas paketas.
        start_idx, end_idx = extract_frame(remaining_samples,
                                         remaining_energy,
                                         add_samples=add_samples,
                                         display=display_extraction,
                                         frame_idx=idx)

        if start_idx == None:
            print(f"Visi paketai ištraukti (kiekis = {idx}).. baigiam.")
```

```

        break

    # Atnaujiname visus kintamuosius ir paslenkame signalą.
    frame = remaining_samples[start_idx:end_idx]
    noise = remaining_samples[:start_idx]

    remaining_samples = remaining_samples[end_idx:]
    remaining_energy = remaining_energy[end_idx:]

    augmented_frames = {} #augment_frames_varying_snr(frame, noise, print_snr=False)
    frame_len = int(400 * fs/int(1e6)) # Atskaičiuojame pagal mikro-sekundes.
    frame_mag = np.abs(frame[:frame_len])

    ### Skaičiuojame signalo amplitudės dispersiją - ieškosime pereinamojo signalo galo.
    variance = calculate_variance(frame_mag,
                                  step=var_step,
                                  width=var_window)

    ### Turime pilną paketą - ištraukiam pereinamojo signalo pradžią. Naudojame dispersiją.
    _, end_idx = get_peak_end_length(variance)

    if end_idx == None:
        continue

    transient_start_idx['variance'] = add_samples
    transient_end_idx['variance'] = end_idx * var_step
    transient = frame[transient_start_idx['variance']:transient_end_idx['variance']]
    real_snr = calculate_snr(np.abs(frame), np.abs(noise))

    if display_frames == True:
        frame_ms = (T * len(frame) * 10e2)

        fig = plt.figure(figsize=(12, 8), layout='constrained')
        gs = fig.add_gridspec(ncols=1, nrows=2)

        ax = fig.add_subplot(gs[0, 0]); ax.grid()
        ax.set(title=f'Ištrauktas {len(frame)} imties taškų paketas (ID = {idx}), trukmė -
{frame_ms:.2f} ms', ylabel="Amplitudė", xlabel="Laikas, ms")
        ax.plot(np.real(frame), label='Reali dalis (I)')
        ax.plot(np.imag(frame), label='Menama dalis (Q)')
        ax.legend(loc='lower right')

        ticks = np.linspace(0, len(frame), 15, dtype=int)
        ax.set_xticks(ticks, labels=['{: .2f}'.format(elem * T * 10e2) for elem in ticks])

    # Paskaiciuojam PSD
    psd = np.abs(np.fft.fftshift(np.fft.fft(frame)))**2
    psd_dB = 10 * np.log10(psd)
    f = np.linspace(fs/-2, fs/2, len(psd))
    ax = fig.add_subplot(gs[1, 0]); ax.grid()

```

```

    ax.set(title=f'Ištraukto {len(frame)} imties taškų paketo signalo galios spektro tankis
(PSD)', ylabel='Amplitudė, dB', xlabel='Dažnis, MHz')
    ax.plot(f/1e6, psd_dB)

if display_transient == True:
    dispersijos_limitas = 50 # len(variance))

    # Vaizduojame dispersijos metodu apskaičiuotą pereinamąjį signalą
    fig = plt.figure(figsize=(10, 4), layout='constrained')
    gs = fig.add_gridspec(ncols=2, nrows=2)

    ax = fig.add_subplot(gs[0, 0])
    ax.set(title=f'Disperija - skaičiavimų žingsnis {var_step} taškai, langas W={var_window}',
           xlabel='Lango numeris, N',
           ylabel=r'Dispersija,  $\sigma^2$ ')
    #ax.axhline(y=np.average(variance))
    ax.axvline(x=end_idx, color='red')
    #ax.set_xlim(0, end_idx + (dispersijos_limitas / var_step))
    ax.plot(variance)

    ax = fig.add_subplot(gs[1, 0])
    ax.set(title='Paketo pradžios amplitudė', xlabel='Imties numeris', ylabel='Amplitudė')
    ax.axvline(x=end_idx * var_step, color='red')
    #ax.set_xlim(0, var_step * end_idx + dispersijos_limitas)
    ax.plot(frame_mag)

    ax = fig.add_subplot(gs[:, 1])
    ax.set(title=f'Išgautas pereinamasis signalas, trukmė - {(len(transient)*T*10e5):.2f}  $\mu$ s',
           xlabel='Imties numeris',
           ylabel='Amplitudė')
    ax.set_xlim(0, len(transient))
    #ax.plot(normalize_minmax_real(np.abs(transient), range=(0, 1)))
    ax.plot(np.abs(transient))
    #ax_top = ax.twinx()
    #ax_top.set_xticks(ax.get_xticks())
    #ax_top.set_xticklabels(['{: .1f}'.format(elem * T * 10e5) for elem in ax.get_xticks()])
    #ax_top.set_xlabel('Laikas, us')

    augmented_transients = {}
    for i, aug_fr in augmented_frames.items():
        augmented_transients[i] =
aug_fr[transient_start_idx['variance']:transient_end_idx['variance']]

    processed_frames.append({'frame': frame, 'transients': { 'original': transient }})
    processed_frames[-1]['transients'].update(augmented_transients)

    idx = idx + 1

del remaining_samples
del remaining_energy

```

```
return processed_frames

for transmitter, modulations in iq_samples.items():
    print(f'Analizuojame siųstuva {transmitter}..')
    training_data[transmitter] = list()
    visualize = False

    for modulation, iq_data in modulations.items():
        extracted_data = extract_frames_and_transients(samples=iq_data,
                                                       fs=fs,
                                                       modulation=modulation,
                                                       display_transient=False,
                                                       display_frames=visualize,
                                                       display_extraction=visualize)

        training_data[transmitter].append({'modulation': modulation, 'data': extracted_data})
    del extracted_data
```

5 priedas. Hilberto-Huango transformacijos spektro skaičiavimas

```
import emd
import os
from PIL import Image

import matplotlib.colors as colors

max_freq = int(80e3) #int(0.2e6)
short_frame_len = 1000

# Y asies formatuotojas.
ticks = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/1e3))

def show_transient_fn(transient_len, frame_short_len, hht_transient, short_frame_time_centres, freq_edges,
hht_frame_short, transient_time_centres, modulation):
    tr_len_x = np.linspace(0, transient_len, 20)
    sh_len_x = np.linspace(0, frame_short_len, 15)
    hht_energy_exists = (hht_transient[:, :transient_len] > 0).astype(int)

    fig = plt.figure(figsize=(12, 8), layout='constrained')
    gs = fig.add_gridspec(ncols=1, nrows=3)

    hht_frame_short = normalize_minmax_real(np.real(hht_frame_short), range=(0, 1))

    ax1 = fig.add_subplot(gs[0, 0])
    im1 = ax1.pcolormesh(short_frame_time_centres, freq_edges, hht_frame_short[:, :frame_short_len],
cmap='gist_heat_r')
    ax1.set(title=f'A) {transmitter.upper()} siųstuvo duomenų paketo pradžios Hilberto - Huango
transformacijos spektras', ylabel='Dažnis, kHz')
    ax1.set_xticks(ticks=sh_len_x, labels=['{:,.2f}'.format(x) for x in (sh_len_x * T * 10e5)])
    ax1.yaxis.set_major_formatter(ticks)
    cb = fig.colorbar(im1); cb.set_label('Amplitudė', rotation=90)

    ax2 = fig.add_subplot(gs[1, 0])
    im2 = ax2.pcolormesh(transient_time_centres, freq_edges, hht_transient[:, :transient_len],
cmap='gist_heat_r', vmin=0, vmax=1)
    ax2.set(title=f'B) Pereinamojo signalo Hilbert - Huang spektras: {lookup_modulation[modulation]}',
ylabel='Dažnis, kHz')
    ax2.set_xlim(0, transient_len)
    ax2.set_xticks(ticks=tr_len_x, labels=['{:,.2f}'.format(x) for x in (tr_len_x * T * 10e5)])
    ax2.yaxis.set_major_formatter(ticks)
    cb = fig.colorbar(im2); cb.set_label('Normalizuota amplitudė', rotation=90)
    ax2.set_xlabel('Laikas, μs')

# Save flags
save_transient = False
save_awgn_transient = False

# Show flags
show_transient = False
```

```

out_dir = 'original'
idx = 0

for transmitter, data in training_data.items():
    main_dir = f'./magistras/{out_dir}/{transmitter}/'
    visuals_dir = main_dir + 'visuals/'

    try:
        os.mkdir(visuals_dir)
    except OSError as error:
        print(f'[ignored]: {error}')

    with open(f'./magistras/{out_dir}/{transmitter}/meta.txt', 'a') as file:
        # Tas pats siustuvas, bet skirtingos moduliacijos.
        for modulation_data in data:
            modulation = modulation_data['modulation']

            # Ciklas per visus paketus..
            for data_frame in modulation_data['data']:
                complex_frame = data_frame['frame']

                # Ciklas per visus vienos paketo pereinamuosius signalus..
                for snr, complex_transient in data_frame['transients'].items():
                    ### Konvertuojame kompleksinį signalą į tik realią dalį.
                    transient = complex_to_real(complex_transient, fs)
                    transient_real = normalize_minmax_real(np.real(transient), range=(-1, 1))
                    transient_len = len(transient_real)

                    if snr == 'original' or snr > 30:
                        continue

                    frame = complex_to_real(complex_frame, fs)
                    frame_real = normalize_minmax_real(np.real(frame), range(-1, 1))
                    frame_len = len(frame_real)

                    frame_short = frame_real[:short_frame_len]
                    frame_short_len = len(frame_short)
                    ###

                    ### Skaičiuojame signalo IMF dedamąsias. Naudojama tik reali signalo dalis.
                    imf_transient = emd.sift.sift(transient_real, max_imfs=4)
                    imf_frame = emd.sift.sift(frame_real, max_imfs=6)
                    imf_frame_short = emd.sift.sift(frame_short, max_imfs=5)

                    imfs_plot = emd.plotting.plot_imfs(imf_frame_short[:fs*3, :])
                    imfs_plot.set(xlabel='Imties numeris, n', ylabel='Amplitudė')
                    imfs_plot.set_title(f'{transmitter.upper()} siustuvas -
{lookup_modulation[modulation]}', fontsize=16)
                    ###

```

```

    ### Skaiciuojame IP/IA/IF
    IP_transient, IF_transient, IA_transient =
emd.spectra.frequency_transform(imf_transient, fs, 'hilbert')
    IP_frame, IF_frame, IA_frame = emd.spectra.frequency_transform(imf_frame, fs,
'hilbert')

    IP_frame_short, IF_frame_short, IA_frame_short =
emd.spectra.frequency_transform(imf_frame_short, fs, 'hilbert')

    transient_time_centres = np.arange(transient_len + 1) - .5
    frame_time_centres = np.arange(frame_len + 1) - .5
    short_frame_time_centres = np.arange(short_frame_len + 1) - .5
    ###

    freq_edges, freq_centres = emd.spectra.define_hist_bins(0, max_freq, 256, 'linear')

    # Analyze the transient of the given IMF's
    f_transient, hht_transient = emd.spectra.hilberthuang(IF_transient, IA_transient,
freq_edges, mode='amplitude', sum_time=False)
    f_frame, hht_frame = emd.spectra.hilberthuang(IF_frame, IA_frame, freq_edges,
mode='amplitude', sum_time=False)
    f_frame_short, hht_frame_short = emd.spectra.hilberthuang(IF_frame_short,
IA_frame_short, freq_edges, mode='amplitude', sum_time=False)

    hht_frame = ndimage.gaussian_filter(hht_frame, 1)
    hht_frame_short = ndimage.gaussian_filter(hht_frame_short, 1)

    """
    Pavaizduojame pereinamąjį signalą ir jo Hilbert - Huang spektrą:
    1. Duomenų paketo priartinta pradžia;
    2. Tik pereinamojo signalo Hilbert - Huang spektra;
    """

    if show_transient:
        show_transient_fn(transient_len, frame_short_len, hht_transient,
            short_frame_time_centres, freq_edges,
            hht_frame_short, transient_time_centres, modulation)

    if save_transient and not save_awgn_transient:
        image_name = f'./magistras/{out_dir}/{transmitter}/data_{idx}'
        file.write(f'{image_name};{lookup_modulation[modulation]}\n')

    # X - axis: transient_time_centres
    # Y - axis: freq_edges
    # Z - axis: hht_transient[:, :transient_len]
    # [rows, columns] = [128, 51]
    # Originalas yra [X, Y], transponuojam, kad gautumėm [X, Y]
    tr = np.flip(hht_transient[:, :transient_len], axis=0)
    tr = normalize_2d_array(tr) # Issaugome jau normalizuotus [0; 1]

    print(f'Generuojamas {tr.shape} paveikslas.. idx = {idx}')

    # Normalizuojame tik atvaizdavimui.

```



```

        im = Image.fromarray((255*tr).astype(np.uint8))
        im.save(visuuls_dir + 'image_{0}.jpeg'.format(idx))
        np.save(main_dir + 'data_' + str(idx), tr)

    if save_awgn_transient and not save_transient:
        image_name = f'./magistras/{out_dir}/{transmitter}/{snr}/data_{idx}'
        file.write(f'{snr};{image_name};{lookup_modulation[modulation]}\n')

        tr = np.flip(hht_transient[:, :transient_len], axis=0)
        tr = normalize_2d_array(tr) # Issaugome jau normalizuotus [0; 1]

        #im = Image.fromarray((255*tr).astype(np.uint8))
        #im.save(visuuls_dir + 'image_{0}dB_{1}.jpeg'.format(snr, idx))
        np.save(main_dir + str(snr) + 'dB/data_' + str(idx), tr)
    idx = idx + 1

```

6 priedas. Gilaus mokymosi pagalbinės funkcijos

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, Subset, DataLoader
from sklearn.model_selection import train_test_split
import pandas as pd
import os
from torch.nn.utils.rnn import pad_sequence
from PIL import Image
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F

def compute_accuracy(model, data_loader, use_probs=False):
    correct_pred, num_examples = 0, 0

    for i, (features, labels) in enumerate(data_loader):

        features = features.to(device)
        labels = labels.to(device)

        out = model(features)
        _, predicted_labels = torch.max(out, dim=1)

        num_examples += labels.size(0)
        correct_pred += (predicted_labels == labels).sum()

    return correct_pred.float() / num_examples * 100

def compute_epoch_loss(model, data_loader):
    curr_loss, num_examples = 0., 0

    with torch.no_grad():
        for features, labels in data_loader:
            features = features.to(device)
            labels = labels.to(device)

            out = model(features)
            loss = F.cross_entropy(out, labels, reduction='sum')

            num_examples += labels.size(0)
            curr_loss += loss

    curr_loss = curr_loss / num_examples
    return curr_loss

class SignalDataset(Dataset):
    def __init__(self, annotations_file, img_dir, num_class, transform=None):
        self.img_labels = pd.read_csv(annotations_file, header=None, delimiter=';')
```

```

self.img_dir = img_dir
self.transform = transform
self.num_class = num_class

def __len__(self):
    return len(self.img_labels)

def __getitem__(self, idx):
    img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
    image = np.load(img_path)
    label = self.img_labels.iloc[idx, 1]

    # Only accept 256x256 max images.
    if self.transform:
        image = self.transform(image)

    return image, torch.tensor(label)

def custom_collate_fn(batch):
    seqs, targets = zip(*batch)
    # max_cols = max(sample.size(1) for sample in seqs)
    max_cols = 256

    # Dirbtinai prapleciame duomenis iki 256 laiko ašyje - pridedame vien tik 0.
    padded_batch = torch.zeros((len(seqs), 1, 256, max_cols), dtype=torch.float32)
    for i, sample in enumerate(seqs):
        _, cols = sample.size()
        padded_batch[i, 0, :256, :cols] = sample

    targets = torch.LongTensor(targets)
    return padded_batch, targets

data_type = 'augmented' # { 'original', 'augmented' }
data = SignalDataset(annotations_file=f'./magistras/{data_type}/dataset/labels.csv',
                    img_dir=f'./magistras/{data_type}/dataset/',
                    num_class=3,
                    transform=torch.from_numpy)

num_features = 16
num_classes = 3

test_sz = 0.2
batch_size = 64
seed = 42

labels = [row[1] for row in data]
train_idx, test_idx, _, _ = train_test_split(range(len(data)),
                                            labels,
                                            stratify=labels,
                                            test_size=test_sz,

```

```
                                random_state=seed
)

train_split = Subset(data, train_idx)
test_split = Subset(data, test_idx)

train_batches = DataLoader(train_split, batch_size=batch_size, shuffle=True, collate_fn=custom_collate_fn)
test_batches = DataLoader(test_split, batch_size=batch_size, collate_fn=custom_collate_fn)

from torchvision import datasets
from torchvision import transforms
from torchsummary import summary

if torch.cuda.is_available():
    device = torch.device("cuda")
    torch.backends.cudnn.deterministic = True

# Device configuration
device = torch.device("cuda")
print('Device:', device)

torch.manual_seed(seed)
```

7 priedas. Gilaus mokymosi modelių sudarymo kodas

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()

        self.conv1 = nn.Sequential(nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride,
padding=1),
                                nn.BatchNorm2d(out_channels),
                                nn.ReLU())

        self.conv2 = nn.Sequential(nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
padding=1),
                                nn.BatchNorm2d(out_channels))

        self.downsample = downsample
        self.relu = nn.ReLU()
        self.out_channels = out_channels

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        if self.downsample:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)
        return out

class FlattenResNet(nn.Module):
    def __init__(self, block, layers, num_classes=3):
        super(FlattenResNet, self).__init__()

        self.inplanes = 64
        self.conv1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=5, stride=1, padding=2),
                                nn.BatchNorm2d(64),
                                nn.ReLU())

        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=4, padding=1)
        self.layer0 = self._make_layer(block, 64, layers[0], stride=1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer2 = self._make_layer(block, 128, layers[2], stride=1)
        self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                nn.Flatten(),
                                nn.Linear(128, num_classes))

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes),
```

```

    )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))
    self.inplanes = planes
    for i in range(1, blocks):
        layers.append(block(self.inplanes, planes))

    return nn.Sequential(*layers)

def forward(self, x):
    x = self.conv1(x)
    x = self.maxpool(x)
    x = self.layer0(x)
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.classifier(x)
    #print(torch.max(F.softmax(x, dim=1), 1))
    return F.softmax(x, dim=1) #, F.softmax(x, dim=1)

summary_model = FlattenResNet(ResidualBlock, layers=[2, 1, 1], num_classes=num_classes).to(device)
summary(summary_model, input_size=(1, 256, 256))
del summary_model

# https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/
# https://github.com/Moddy2024/ResNet-34/blob/main/resnet-34.ipynb

class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=3):
        super(ResNet, self).__init__()

        self.inplanes = 64
        self.conv1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size = 7, stride = 2, padding = 3),
                                   nn.BatchNorm2d(64),
                                   nn.ReLU())

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer0 = self._make_layer(block, 64, layers[0], stride=1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer2 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer3 = self._make_layer(block, 512, layers[3], stride=2)
        self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                       nn.Flatten(),
                                       nn.Linear(512, num_classes))

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes),
            )

```

```

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.classifier(x)
        return x

summary_model = ResNet(ResidualBlock, layers=[3, 4, 6, 3], num_classes=num_classes).to(device)
summary(summary_model, input_size=(1, 256, 256))
del summary_model

class VGG16(nn.Module):
    def __init__(self, num_features, num_classes):
        super(VGG16, self).__init__()
        # Padding: (1(32-1)- 32 + 3)/2 = 1
        # Input: 256xN
        self.conv1 = nn.Sequential(nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(3, 3),
stride=(1,1), padding=1),
                                nn.ReLU(),
                                nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3, 3),
stride=(1, 1), padding=1),
                                nn.ReLU(),
                                nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

        self.conv2 = nn.Sequential(nn.Conv2d(in_channels=64, out_channels=128, kernel_size=(3, 3),
stride=(1, 1), padding=1),
                                nn.ReLU(),
                                nn.Conv2d(in_channels=128, out_channels=128, kernel_size=(3, 3),
stride=(1, 1), padding=1),
                                nn.ReLU(),
                                nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

        self.conv3 = nn.Sequential(nn.Conv2d(in_channels=128, out_channels=256, kernel_size=(3, 3),
stride=(1, 1), padding=1),
                                nn.ReLU(),
                                nn.Conv2d(in_channels=256, out_channels=256, kernel_size=(3, 3),
stride=(1, 1), padding=1),
                                nn.ReLU(),

```

```

        nn.Conv2d(in_channels=256, out_channels=256, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

    self.conv4 = nn.Sequential(nn.Conv2d(in_channels=256, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

    self.conv5 = nn.Sequential(nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(3, 3),
stride=(1, 1), padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

    self.classifier = nn.Sequential(nn.Linear(512, 4096),
        nn.ReLU(True),
        #nn.Dropout(p=0.5),
        nn.Linear(4096, 4096),
        nn.ReLU(True),
        #nn.Dropout(p=0.5),
        nn.Linear(4096, num_classes))

    for m in self.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlinearity='relu')
            if m.bias is not None:
                m.bias.detach().zero_()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)

        # https://github.com/keras-team/keras/issues/1920
        avg_pool = nn.AdaptiveAvgPool2d((1, 1))

```



```
x = avg_pool(x)
x = x.view(x.size(0), -1)
x = self.classifier(x)
return x

summary_model = VGG16(num_features=num_features, num_classes=num_classes).to(device)
summary(summary_model, input_size=(1, 256, 256))
del summary_model
```

8 priedas. Gilaus mokymosi struktūrinis kodas

```
import time
import torch.nn.functional as F
import gc

learning_rate_adam = 0.0001
learning_rate = 0.001
num_epochs = 20

# Permokymas VGG16 ir Adam su 0.001
#model = VGG16(num_features=num_features, num_classes=num_classes).to(device)
#model = ResNet(ResidualBlock, layers=[3, 4, 6, 3], num_classes=num_classes).to(device)
#model = FlattenResNet(ResidualBlock, layers=[2, 1, 1], num_classes=num_classes).to(device)
model = SimpleCnn(num_classes=num_classes).to(device)

#optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
#optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay = 0.001, momentum = 0.9)
early_stopper = EarlyStopper(patience=3, min_delta=0.2)
criterion = F.cross_entropy

start_time = time.time()

epoch_test_loss = list()
epoch_train_loss = list()
epoch_acc = list()
epoch_test_acc = list()

for epoch in range(num_epochs):

    model.train()
    for batch_idx, (features, labels) in enumerate(train_batches):
        features = features.to(device)
        labels = labels.to(device)

        out = model(features)
        cost = criterion(out, labels)
        optimizer.zero_grad()

        cost.backward()
        optimizer.step()

        del features, labels, out
        torch.cuda.empty_cache()
        gc.collect()

        if not batch_idx % 10:
            print('Epoch: %03d/%03d | Batch %04d/%04d | Cost: %.4f' % (
                epoch+1, num_epochs, batch_idx, len(train_batches), cost)
            )
```

```
model.eval()
# with torch.set_grad_enabled(False):
with torch.inference_mode():
    train_acc = compute_accuracy(model, train_batches, False).cpu()
    test_acc = compute_accuracy(model, test_batches, False).cpu()

    train_loss = compute_epoch_loss(model, train_batches).cpu()
    test_loss = compute_epoch_loss(model, test_batches).cpu()

    epoch_acc.append(train_acc)
    epoch_test_acc.append(test_acc)
    epoch_train_loss.append(train_loss)
    epoch_test_loss.append(test_loss)

    print('Epoch: %03d/%03d | Train: %.3f% | Test: %.3f% | Train Loss: %.3f | Test Loss: %.3f' %
          (epoch+1, num_epochs, train_acc, test_acc, train_loss, test_loss))

    print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))
print('=> Total Training Time: %.2f min' % ((time.time() - start_time)/60))
```